PROCEDIMIENTO PARA MIGRAR APLICACIONES WEB DE ASP.NET WEBFORMS A ASP.NET MVC

JUAN CAMILO DÍAZ GARCÍA

ESCUELA DE INGENIERÍA DE ANTIOQUIA
INGENIERÍA INFORMÁTICA
ENVIGADO
2012

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

PROCEDIMIENTO PARA MIGRAR APLICACIONES WEB DE ASP.NET WEBFORMS A ASP.NET MVC

JUAN CAMILO DÍAZ GARCÍA

Trabajo de grado para optar al título de Ingeniero Informático

Ingeniero Alejandro Arroyave Buriticá

Hexacta S.A (Argentina)

ESCUELA DE INGENIERÍA DE ANTIOQUIA
INGENIERÍA INFORMÁTICA
ENVIGADO
2012

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

“Este trabajo esta dedicado a Dios antes que todo, a mi papá, mi mamá y mi hermano que tanto me ha apoyado durante estos años de desarrollo profesional y a mis dos mejores amigos Giovani y Laura por tanto momentos inolvidables en estos últimos años”

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

AGRADECIMIENTOS

Agradezco primero a Dios que ilumina mi mente, a la virgen, a San Judas Tadeo por todos los favores recibidos, a mi familia, a mi tutor Alejandro Arroyave por toda su ayuda, acompañamiento, asesoría y paciencia en el desarrollo de este trabajo, a la Escuela de Ingeniería de Antioquia, en especial al Carlos Jaime Noreña por toda asesoría y paciencia en el desarrollo del anteproyecto y proyecto de grado.

A Coltejer S.A., mas específicamente a la fundación Coltejer por el apoyo económico y moral durante estos años de desarrollo profesional, especialmente a la señora Teresa Perea encargada de la fundación y a la Dr. Elsa Gladys Muñoz Gutiérrez. Agradezco también a la Dr. Alicia Osorio Builes, gerente de tecnología e informática de Coltejer por el apoyo que me ha dado para el desarrollo de mis estudios y a mis compañeros y amigos.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

CONTENIDO

pág.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

LISTA DE TABLAS

Pág.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

LISTA DE FIGURAS

Pág.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

LISTA DE ANEXOS

pág.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

GLOSARIO

AJAX: Asynchronous JavaScript And XML (JavaScript asíncrono y XML). Es una técnica de desarrollo utilizada para mantener una comunicación asíncrona con el servidor en segundo plano.

ASP: Active Server Pages, también conocido como ASP clásico es un lenguaje de lado de servidor para crear página dinámicas

CIL: Common Intermediate Language (lenguaje intermedio común), hace parte de .NET Framework y es un lenguaje de programación legible por humanos de mas bajo nivel que CLI. Todos los lenguajes soportados por .NET Framework se compilan a CIL.

CLI: Common Language Infraestructure (infraestructura de lenguaje común), es un componente de .NET Framework que permite a través un de entorno virtual que las aplicaciones escritas en distintos lenguajes de alto nivel puedan ejecutarse en diferentes plataformas, sin rescribir el código.

CLR: Common Language Runtime (entorno de ejecución de lenguaje común). Entorno de ejecución de los programas que corren bajo la plataforma .NET.

C#: pronunciado C Sharp, es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

COOKIE: fragmento de información que se almacena en el disco duro de un visitante de una página web a petición del servidor.

EDM: Entity Data Model, herramienta de .NET que sirve para especificar un modelo conceptual de los datos.

Framework: es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la que otro proyecto de software puede ser más fácilmente organizado y desarrollado

GET: es un método HTTP que sirve para pasar datos de una página a otra o para capturar datos ingresados por los usuarios en formularios, extrayendo los datos a través de la URL

GNU/Linux: es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux con el sistema operativo GNU

GUI: Graphic User Interface (interfaz gráfica de usuario)

HTML: Hypertext Markup Language (lenguaje de marcado de hipertexto)

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto), es un protocolo usado para las transacciones web.

IIS: Internet Information Services, es el servidor web de los sistemas operativos Windows.

IntelliSense: Microsoft IntelliSense es la aplicación de autocompletar, mejor conocido por su utilización en Microsoft Visual Studio.

JavaScript: es un lenguaje de programación utilizado en el desarrollo web. Se define como orientado a objetos, basado en prototipos, imperativo y dinámico

JQuery: es un Framework para JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web

Jscript: es la implementación de Microsoft del estándar ECMAScript (JavaScript también es una implementación de ECMAScript.)

JSON: JavaScript Object Notation (notación de objetos JavaScript), es un formato ligero para el intercambio de datos que no requiere el uso de XML.

LINQ: Language Integrated Query, es un proyecto de Microsoft que agrega consultas nativas semejantes a las de SQL a los lenguajes de la plataforma .NET

Metadatos: es un término que se refiere a datos sobre los propios datos.

MVC: Modelo Vista Controlador

POST: es un método HTTP que sirve para pasar datos de una página a otra o para capturar datos ingresados por los usuarios en formularios, a diferencia del método GET donde los datos están contenidos en la URL, en el método POST se transmite la información en payload, es decir la información transmitida no es visible para el usuario.

POSTBACK: es una llamada HTTP-POST a la misma página donde se encuentra un formulario activo

RAD: Rapid Application Development (desarrollo rápido de aplicaciones).

Script: secuencia de comandos, es un programa usualmente simple, que se almacena en un archivo de texto plano

TDD: Test-Driven Development (desarrollo guiado por pruebas).

UI: User Interface (interfaz de usuario).

URL: Uniform Resource Locator (localizador de recursos uniforme)

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

VBscript: Visual Basic Script Edition, es un lenguaje interpretado por el Windows Scripting Host de Microsoft. Su sintaxis es una variación del lenguaje de programación Visual Basic

View State: es un método que ASP.NET usa para conservar los datos de la página y los valores de control cuando se envía información desde y hacia el servidor, esta información por lo general esta puesta en campos ocultos.

Visual Basic .NET: es un lenguaje de programación orientado a objetos que se puede considerar una evolución de Visual Basic implementada sobre el framework .NET

XML: Extensible Markup Language (lenguaje de marcas extensible), permite definir la gramática de lenguajes específicos para estructurar documentos grandes y bases de datos.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

RESUMEN

En la actualidad el marco de trabajo .NET de Microsoft provee dos modelos de desarrollo para entornos web: ASP.NET Web Forms y ASP.NET MVC. El primero fue creado hace aproximadamente diez años, cuando el desarrollo de aplicaciones web era aún incipiente; este fue pensado para que el desarrollo web fuera lo más parecido posible al desarrollo de aplicaciones de escritorio, teniendo como propósito que fuera orientado a eventos y permitiera el desarrollo rápido de aplicaciones. Este modelo de desarrollo es un verdadero éxito, y gracias a ello se ha mantenido durante tanto tiempo. Pero su estructura también tiene una serie de limitantes como la dificultad de realizar pruebas, la combinación de lógica de presentación con lógica de la aplicación y la falta de control sobre los códigos HTML, CSS, y JavaScript generados, hacen difícil el mantenimiento, la localización y corrección de errores, las pruebas, la adición de nuevas funcionalidades y en ocasiones se generan páginas web muy pesadas, que hacen disminuir el desempeño del sitio.

ASP.NET MVC representa una nueva alternativa para el desarrollo web y una solución a las limitantes de Web Forms; también surge como una respuesta a lo que el desarrollo de aplicaciones web requiere hoy en día. ASP.NET MVC permite hacer una clara separación de responsabilidades (presentación, acceso a fuentes de datos y la lógica de la aplicación), el desarrollo guiado por pruebas, la reutilización de los componentes, la simplicidad en el desarrollo, el control sobre el código HTML generado y el mantenimiento, lo que lleva a que las aplicaciones sean mucho más eficientes y a futuro más escalables

En la actualidad existen miles de aplicaciones desarrolladas en Web Forms, algunas de ellas debido a las limitaciones de este modelo de desarrollo se ven en la necesidad de ser transformadas al modelo ASP.NET MVC. Este trabajo presenta entonces una herramienta de ayuda a esta necesidad; un procedimiento de migración para convertir aplicaciones de Web Forms a MVC, haciendo un análisis de la estructura y forma de trabajar de cada modelo de desarrollo, examinando sus similitudes y diferencias e implementando el procedimiento en un caso práctico.

Palabras clave: .NET Framework, ASP.NET Web Forms, ASP.NET MVC, migración, aplicaciones web, desarrollo web.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

ABSTRACT

Currently, Microsoft NET framework provides two development models for web environments: ASP.NET Web Forms and ASP.NET MVC. The first one was created about ten years ago, when the web development was incipient; this was designed thinking to make as similar as possible the web development and the desktop applications development: event-driven development and allow rapid application development. This development model was quite successful; therefore it has persisted for so long. But the structure also has a number of limitations such as the difficulty of testing, the combination of presentation logic with business logic and lack of control over HTML, CSS, and JavaScript generated, it make difficult to maintain, finding and correcting errors, the testing process, adding new features, and sometimes web pages generated are very heavy then it decrease the performance of the site.

ASP.NET MVC is a new alternative for web development and a solution to the limitations of Web Forms, also it is a response about what the web application development needs today. ASP.NET MVC allows a clear separation of responsibilities (presentation, access to data sources and business logic), test-driven development, reusing components, simplicity in the development, control over the generated HTML code and maintenance, leading to applications much more efficient and more scalable in the future.

There are thousands of applications developed in Web Forms; some of them, due to the limitations of this model of development, need to be migrated to ASP.NET MVC model. This paper presents a tool that helps satisfying this need: a migration procedure to convert Web Forms applications to MVC, with analysis of the structure and working methods of each development model, examining their similarities and differences and implementing the procedure in a case study.

Key words: .NET Framework, ASP.NET Web Forms, ASP.NET MVC, migration, web applications, web development.

La información presentada en este documento es de exclusiva responsabilidad de los autores y no compromete a la EIA

INTRODUCCIÓN

Este documento describe un procedimiento de migración de aplicaciones ASP.NET Web Forms a ASP.NET MVC.

En la primera parte se hace un estudio de las principales características del modelo de desarrollo Web Forms, como lo son el patrón de diseño en que se basa, el ciclo de vida de la página, el acceso a fuentes de datos, la forma de construir la lógica de negocio, la presentación de la información y la seguridad.

De la misma forma se miran los aspectos centrales del marco de trabajo ASP.NET MVC, estudiando especialmente cada parte de la separación de conceptos que este hace: el modelo, el controlador y la vista. Aunque también se evalúan aspectos como la seguridad y los patrones de diseño que utiliza.

Luego se elabora un comparativo de ambos modelos de desarrollo, permitiendo encontrar las similitudes y diferencias entre ellos, analizando las ventajas y desventajas que poseen.

Finalmente se hace una exploración y análisis de metodologías de migración, donde se explica el caso a migrar, se procede a realizar la construcción del escenario de migración y se valida que el propósito funcional de la aplicación no haya sido afectado por medio de pruebas unitarias.

1. PRELIMINARES

1.1 PLANTEAMIENTO DEL PROBLEMA

El Framework .NET de Microsoft está diseñado para permitir el desarrollo rápido de aplicaciones tanto en Windows como en la web, por eso cuando se lanzó ASP.NET en el año 2002, Microsoft pretendía minimizar la curva de aprendizaje de aquellos desarrolladores acostumbrados a programar para ambientes de escritorio de forma que pudieran implementar aplicaciones para la web de forma similar al desarrollo de aplicaciones para Windows. (López Ramírez, 2009)

ASP.NET Web Forms fue un desarrollo importante que impulsó la implementación de soluciones web y servicios web a nivel mundial, pero con el tiempo surgieron inconvenientes relacionados con su forma de trabajar. Por ejemplo: las aplicaciones construidas en Web Forms son orientadas a eventos y realizan las peticiones al servidor principalmente por medio del método Postback. Además de utilizar otro mecanismo llamado View State para almacenar datos y los estados de los controles de la página (MSDN Library, 2011); a medida que los desarrollos crecen, el volumen de datos que viaja en el ViewState desde y hacia el servidor se hace más grande, los tiempos de respuesta se hacen más largos. Otro problema, es la fuerza con que crecieron los navegadores web en los últimos años (especialmente aquellos referentes a los dispositivos móviles), ya que el código generado por los controles de servidor ASP.NET Web Forms no respetan los estándares de HTML al combinar el código con CSS, lo que genera múltiples problemas de compatibilidad con estos y dificulta la manipulación del código HTML con JavaScript. Además de las dificultades para agregar nuevas funcionalidades sobre una aplicación ya construida, debido a que puede que no se encuentren separadas las responsabilidades (acceso a fuentes de datos, lógica de la aplicación y presentación), aunque se depende del estilo de programación utilizado..

En respuesta a estos problemas y a la acogida que tiene el patrón Modelo Vista Control en el desarrollo de soluciones web, especialmente en leguajes como PHP, es lanzado en el año 2009 ASP.NET MVC: Este permite hacer una clara separación de responsabilidades, entre la interfaz, la lógica de negocio y el control, lo cual permite el desarrollo guiado por pruebas (TDD) (Sanderson, 2010). Además de hacer posible el control sobre el código HTML generado y el mantenimiento, lo que lleva a que las aplicaciones sean mucho más eficientes y a futuro más escalables (Aguilar, 2010)

En la actualidad, muchas aplicaciones de la industria que fueron construidas en ASP.NET Web Forms tienen la necesidad de agregar nuevas funcionalidades y tener mejores tiempos de respuesta en el servidor pero la forma como fueron implementadas se los impide, ya son desarrollos demasiado complejos. Una solución a esto es migrarlas a ASP.NET MVC, aunque sea un proceso largo, que involucra un cambio total en la lógica de la aplicación, pero que constituye una mejora importante en el rendimiento de la aplicación y le brinda mejores posibilidades de adaptación a cambios futuros sin perder la

calidad esperada. Por lo que se hace necesario documentar un procedimiento que sirva como guía para realizar este proceso.

1.2 OBJETIVOS DEL PROYECTO

1.2.1 Objetivo General:

Documentar un procedimiento que permita la migración de aplicaciones web desarrolladas en ASP.NET Web Forms a ASP.NET MVC.

1.2.2 Objetivos Específicos:

- Identificar el modelo de ciclo de vida de las páginas implementadas en ASP.NET Web Forms.
- Analizar la implementación del patrón Modelo Vista Control en ASP.NET MVC.
- Detectar equivalencias entre la estructura de aplicaciones de ASP.NET Web Forms y la de ASP.NET MVC.
- Desarrollar un caso de migración ente ASP.NET Web Forms y ASP.NET MVC, documentarlo y validar la migración de datos.

1.3 MARCO DE REFERENCIA

1.3.1 Desarrollo Web

Desarrollo web es un término amplio para el proceso de escribir una página o sitio web. Las páginas web se escriben con HTML, CSS y JavaScript. Estas páginas pueden consistir en texto y gráficos sencillos que se asemejan a un documento. Las páginas también pueden ser interactivas o mostrar información que cambia con el tiempo. Las páginas de servidor interactivas son un poco más complejas de escribir, pero habilitan sitios web más enriquecidos. Actualmente, la mayoría de las páginas son interactivas y proporcionan servicios en línea modernos como carros de compras, visualizaciones dinámicas e incluso redes sociales complejas. (MSDN, 2012)

El navegador siempre va a ser el intérprete de las páginas web, a este solo le llega código HTML, CSS, JavaScript. Lo que en el desarrollo web se llama lenguaje cliente. HTML se encarga del contenido de la página, CSS de los estilos de presentación y JavaScript se encarga mejorar la experiencia del usuario al utilizar la página haciendo el contenido estático HTML en contenido dinámico a medida que el usuario interactúa con la página.

Existe otro lenguaje que es el lenguaje de servidor el cual se ejecuta cuando un usuario desde su navegador hace una petición de información. El lenguaje de servidor es el que hace que el contenido de la página sea dinámico, permitiendo también guardar o

consultar información almacenada en bases de datos, o que se realice una función específica dependiendo de la petición del usuario. Este código se ejecuta solo en el servidor y tiene como salida el código HTML que será enviado al navegador como respuesta al usuario. Este trabajo trata sobre el lenguaje servidor ASP.NET, parte del Microsoft .Net Framework, en dos modelos o patrones de programación Web Forms y MVC. Estos dos modelos serán ampliados más en detalle en el desarrollo de este trabajo.

1.3.2 Microsoft .NET Framework

También conocido solo como .NET Framework, es un marco de trabajo desarrollado por Microsoft que se ejecuta principalmente en ambientes Windows, se puede definir como una colección de herramientas, tecnologías y lenguajes de programación que trabajan juntos para proveer las soluciones necesarias que permitan desarrollar aplicaciones empresariales robustas y confiables. Esta plataforma proporciona gran flexibilidad ya que permite construir diferentes tipos de aplicaciones, principalmente aplicaciones Windows de escritorio, servicios web distribuidos y aplicaciones web, utilizando diferentes lenguajes de programación. (Wiley Online Library, 2012)

Este marco de trabajo surgió del problema de tener multiples entornos de desarrollo independientes, como lo son: el desarrollo para aplicaciones de escritorio, aplicaciones web, servicios web, entre otros. Por esto Microsoft desarrolló un entorno que es un entorno unificado, un marco de trabajo independiente del lenguaje de programación que provee herramientas y soliciones predefinidas que ayudan a construir aplicaciones robustas, siempre teniendo en mente el ambiente de Internet. Fue lanzado oficialmente en el año 2002 con la versión 1.1, diez años después su última version es la 4.5, lanzada en agosto de 2012.

.NET Framework es un componente de Windows que además de proveer un entorno de desarrollo soporta la compilación y ejecución de aplicaciones y de servicios web XML. El marco de trabajo esta diseñado para cumplir principalmente con los siguientes objetivos:

- Proveer un ambiente de desarrollo orientado a objetos, donde el código puede ser ejecutado localmente, remotamente o distribuido en internet.

- Suministrar un ambiente de ejecución que minimice los errores en la implantación y los conflictos de versiones.

- Ofrecer un entorno de ejecución de código que promueva la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.

- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.

- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.

- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se pueda integrar con otros tipos de código. (MSDN Library, 2012)

.NET Framework esta consituido básicamente por dos componentes principales: El entorno en tiempo de ejecución de lenguaje común (CLR - Common Language Runtime) y la biblioteca de clases de .NET Framework (BCL – Base Class Library). En la figura 1, podemos observar la estructura general de .NET Framework

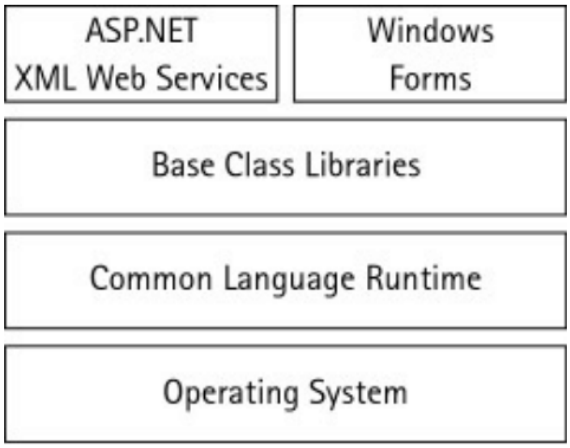

Figura 1 - Estructura de .NET Framework

Según la figura 1, acerca de la estructura de .NET Framework podemos ver que este está compuesto por cuatro capas. La primera tiene que ver con el sistema operativo: .NET corre sobre sistemas operativos tipo Windows, tanto las versiones para PC como para servidores (Windows Server), aunque también es posible que .NET funcione bajo otros sistemas operativos como lo hace el proyecto Mono, un grupo de herramientas libres basadas en GNU/Linux y compatibles con .NET. En la siguiente capa esta el CLR (Common Language Runtime) que es el motor que administra la ejecución del código y uno de los componentes más importantes de .NET y del que hablaremos más adelante. Luego está la biblioteca de clases de .NET que provee una serie de clases predefinidas y herramientas para facilitar el desarrollo de todo tipo de aplicaciones. Y finalmente en la ultima capa, la capa superior, están ASP.NET y Windows Forms, los dos grandes mundos de desarrollo en que podemos dividir .NET Framework; el primero e posibilita el desarrollo de aplicaciones web y la creación de servicios web, y el segundo, se encarga de los desarrollos para aplicaciones de escritorio y aplicaciones de consola.

La principal ventaja del entorno de desarrollo de .NET, es que ayuda en gran medida a acelerar la productividad durante el desarrollo, principalmente porque los programadores tienen la posibilidad de seleccionar el lenguaje de programación de su preferencia, para obtener el mismo resultado. El único requisito que tiene es que el lenguaje debe ser uno orientado a objetos. Microsoft provee básicamente cuatro lenguajes junto con .NET: Visual Basic .NET, C#, C++ .NET y JScript, aunque es posible agregar otros diferentes, siempre que se descarguen e instalen en el entorno (MSDN Library, 2012). La flexibilidad del uso de lenguajes de programación, sumada a la biblioteca de clases y la posibilidad

de obtener otros componentes desarrollados por terceros, hacen de .NET una de las mejores alternativas en el mercado para el desarrollo de software.

Microsoft provee un entorno de desarrollo integrado (IDE – Integrated Development Enviroment) para .NET, se trata de Microsoft Visual Studio; este IDE proporciona todas las herramientas para hacer el desarrollo mucho mas fácil e intuitivo, ya que ayuda a la creación, compilación y publicación de los distintos tipos de desarrollos soportados por .NET Framework.

A continuación vamos a describir los dos principales componentes de .NET Framework: el common language runtime o entorno en tiempo de ejecución de lenguaje común (más conocido como CLR por sus siglas en inglés) y la biblioteca de clases de .NET Framework.

- **Common Language Runtime (CLR)**

El componente más importante de .NET Framework es el common language runtime o entorno en tiempo de ejecución de lenguaje común (CLR). El CLR gestiona y ejecuta el código escrito en cualquiera de los lenguajes de programación soportados por .NET y proporciona acceso a una variedad de servicios que hacen el proceso de desarrollo mucho más fácil. CLR es en si la base de la arquitectura .NET, es el motor que controla la ejecución del código.

Este componente funciona de una manera similar a la maquina virtual de Java, el CLR activa los objetos, se encarga de verificar la seguridad de ellos, gestiona la memoria que estos van a utilizar, los ejecuta y se encarga de limpiar la memoria cuando estos no se utilizan mas. (Thai & Lam, 2002)

El CLR ha sido desarrollado para ser muy superior a anteriores entornos de tiempo de ejecución, como VB runtime, todo para lograr los siguientes objetivos:

- La integración de diferentes lenguajes de programación
- Seguridad en el acceso al código
- Gestión del ciclo de vida de los objetos
- Administración de memoria
- Depuración y soporte
- Manejo de excepciones
- Compilación
- Ensamblados
- Administración de hilos

- Recolector de basura

El código compilado dentro del CLR es llamado código administrado. El código administrado provee los metadatos que el CLR necesita para suministrar el soporte a los diferentes lenguajes de programación, la seguridad, la gestión del ciclo de vida de los objetos y la administración de memoria. (Wiley Online Library, 2012)

El CLR en términos generales es el encargado descompilar una forma de código intermedio llamada Common Intermediate Language (CIL, anteriormente conocido como MSIL, por Microsoft Intermediate Language), al código de maquina nativo, mediante un compilador en tiempo de ejecución. (Wikipedia, 2012) No debe confundirse el termino CLR con el de máquina virtual, ya que una vez que el código esta compilado, corre nativamente sin intervención de una capa de abstracción sobre el hardware subyacente.

El proceso más importante del CLR es la compilación. En resumen, primero el CLR convierte el código .NET a código CIL (Common Intermediate Language – lenguaje intermedio común). Este código puede ser ejecutado desde cualquier máquina y después el código CIL es trasformado al lenguaje nativo de la máquina en aquella donde se ejecute. El ejemplo mas claro lo tenemos con una aplicación ASP.NET: cuando se compila, se genera el código CIL y cuando ejecutamos la página por primera vez, observamos que esta tarda mas de lo normal en ser cargada; esto es porque se esta generando el código en lenguaje de máquina, se esta ejecutando el JIT, pero ya después de la primera ejecución la página se ejecuta sin ningún contratiempo.

- **Biblioteca de clases de .NET Framework (BCL - Base Class Library)**

La biblioteca de clases de .NET (BCL) es una colección de clases e interfaces integradas con el CLR que proporcionan acceso a una serie de utilidades especificas y a diferentes tipos de funciones que ayudan a hacer más fácil el desarrollo de software. (Wiley Online Library, 2012) La biblioteca de clases es orientada a objetos, provee unas clases genéricas que permiten dar una gran funcionalidad a cualquier solución de una forma sencilla y rápida. BCL no solo hace mas fácil desarrollar sobre .NET Framework, también reduce el tiempo asociado con el aprendizaje de nuevas características de .NET Framework y adicionalmente componentes de terceros pueden integrarse sin problemas con las clases de .NET Framework. (MSDN Library, 2012)

Por ejemplo, La colección de clases de .NET Framework implementa unas interfaces que se pueden usar para desarrollar nuestra propia colección de clases, esta colección de clases personalizadas se integrara a la perfección con la biblioteca de clases de .NET.

Tal como se puede esperar de una biblioteca de clases orientada a objetos, BCL permite realizar una variedad de tareas de programación, entre las que se encuentran el manejo de cadenas de caracteres (string), arreglos de datos, conexión a las bases de datos, acceso y edición de archivos, entre otras. Adicional a estas tareas comunes BLC incluye el soporte a diferentes ambientes de desarrollo especializados. Por ejemplo, se puede usar .NET Framework para desarrollar los siguientes tipos de aplicaciones y servicios: (MSDN Library, 2012)

- Aplicaciones de Consola
- Aplicaciones de interfaz Windows
- Aplicaciones WPF (Windows Presentation Foundation)
- Aplicaciones web ASP.NET
- Windows services
- Aplicaciones orientas a servicios usando WCF (Windows Communication Services)
- Flujos de trabajo usando WF (Windows Workflow Foundation)

1.3.3 ASP.NET

ASP.NET es un marco de trabajo (Framework) para el desarrollo de aplicaciones web de propiedad de Microsoft, provee todos los recursos para desarrollar aplicaciones empresariales basadas en la web, permitiendo crear sitios web dinámicos, aplicaciones web y servicios web XML. Fue lanzada en enero de 2002 con la versión 1.1 de .NET Framework y es la tecnología sucesora de ASP (Active Server Pages), al hacer parte de .NET Framework se construye y ejecuta sobre el CLR, permitiendo a los programadores escribir código ASP.NET en cualquier lenguaje soportado por .NET. Además por poseer todas las características de .NET Framework, facilita el desarrollo rápido de aplicaciones, la realización de pruebas, la flexibilidad y la escalabilidad. (Wikipedia, 2012)

Microsoft ASP.NET es específicamente una tecnología del lado del servidor utilizada para crear aplicaciones web dinámicas e interactivas. Una página ASP.NET es un archivo HTML que contiene controles de servidor que son procesados por el servidor web antes de ser enviados al navegador del usuario. Es posible combinar ASP.NET con XML y HTML para crear poderosos sitios web.

Las aplicaciones web hechas bajo la arquitectura ASP.NET funcionan de la siguiente forma: Todos los clientes web o navegadores, se comunican a través de Internet Information Services (IIS); ASP.NET solo funciona sobre IIS, este servicio debe estar instalado y habilitado en el sistema operativo del servidor web, IIS descifra las credenciales del cliente y opcionalmente lo autentica; en el caso de IIS se encuentre configurado para permitir usuarios anónimos, la autenticación no se realiza, en caso contrario si se realiza la autenticación. Para un mejor entendimiento ver la figura 2.

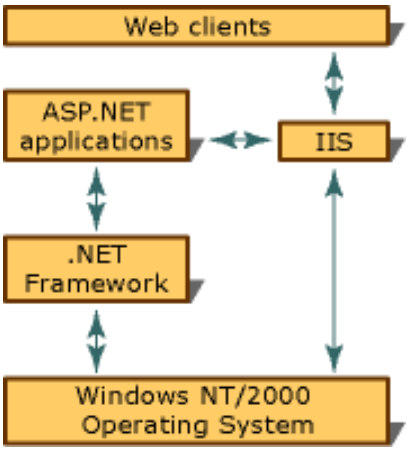

Figura 2 - Arquitectura ASP.NET

Como se puede ver en la figura 2, IIS se encarga también de comunicarse con las aplicaciones ASP.NET y si el cliente se encuentra autorizado para obtener el servicio, devuelve al IIS el recurso que es solicitado; valiéndose de las herramientas que le proporciona .NET Framework. (MSDN Library, 2012)

ASP.NET posee tres modelos o estructuras de programación distintas para construir aplicaciones web dinamicas, según el sitio oficial de Microsoft sobre ASP.NET. Estos tres modelos son: ASP.NET Web Forms, ASP.NET MVC y ASP.NET Web Pages. Este trabajo trata especificamente solo los dos primeros modelos, el tercero ASP.NET Web Pages no será tratado en este trabajo, ya que no es una tecnología muy usada y se utiliza básicamente para proyectos cortos y rápidos.

- **ASP.NET Web Forms**

ASP.NET Web Forms es uno de los tres diferentes modelos de programación que se pueden usar para crear aplicaciones web ASP.NET. Como cualquier recurso en la web, en Web Forms, los usuarios hacen una petición al servidor a través de un navegador y este les devuelve un formulario con la interfaz gráfica (UI) con la que el usuario puede interactuar. Esta interfaz gráfica es un código HTML construido a partir de una combinación de HTML, controles de servidor y lógica de negocio (código de servidor). (ASP.NET, 2012)

ASP.NET Web Forms permite crear páginas Web dinámicas con un modelo orientado a eventos, utilizando una serie de controles y componentes que provee el ambiente de desarrollo, algunos de ellos son tan simples y fáciles de usar que solo se necesita arrastrarlos para usarlos. Estos controles le posibilitan al desarrollador el desarrollo rápido de aplicaciones sofisticadas con una potente interfaz de usuario y acceso a bases de datos.

Web Forms es similar a Windows Forms en la manera en que proporciona las propiedades, métodos y eventos para los controles que tiene disponibles. Sin embargo, estos elementos de interfaz de usuario se hacen en el lenguaje más pertinente dependiendo de la solicitud, por ejemplo, HTML. (MSDN Library, 2012)

Cuando se pensó en Web Forms, Microsoft tenía en mente que para el desarrollador fuera fácil realizar la transición entre el desarrollo de aplicaciones para ambientes de escritorio y el desarrollo web. Es por esto que en Web Forms, podemos encontrar gran variedad de controles y ayudas muy similares a los usados en el desarrollo de aplicaciones de escritorio, como los textbox, labels, datagrids, etc. Web Forms entonces le permite al desarrollador a través del IDE (Visual Studio) seleccionar y arrastrar los controles de servidor para diseñar la página Web Forms, para así poder fácilmente asignar propiedades, métodos y eventos a cada control de la página y la página misma que definan el comportamiento de esta dependiendo de su interacción con el usuario.

ASP.NET Web Forms contiene las siguientes características:

- Separación de código de interfaz de usuario HTML y la lógica de aplicación
- Una completa colección de controles de servidor para las tareas más comunes, incluyendo acceso a bases de datos.
- Poderosos enlaces de datos
- Al estar basada en eventos modelo de programación que es familiar para programadores en Visual Basic
- Soporte para usar Ajax, en caso de no dominar muy bien Javascript
- Permitir a terceros crear controles que proporcionan funcionalidades adicionales

Las aplicaciones desarrolladas bajo el modelo de programación ASP.NET Web Forms basan su funcionamiento teóricamente en el patrón de diseño Page Controller el que será explicado con más detalle en el desarrollo de este trabajo.

Cuando el controlador de la página recibe una solicitud, extrae todos los datos pertinentes e invoca a las actualizaciones del modelo, y envía la solicitud a la vista. La vista, a su vez depende del modelo para la recuperación de los datos que se muestran. Definiendo un controlador que aísle el modelo de los detalles de la solicitud Web; por ejemplo, usando parámetros de consulta o campos de formulario ocultos para pasar parámetros a la página. En esta forma, se crea un controlador para cada eslabón de la aplicación Web. Esto mantiene los controladores simples, ya que sólo tienen que preocuparse por una acción a la vez.

La creación de un controlador independiente para cada página web o acción puede conducir a la duplicación de código. Por lo tanto, debe crear una clase base común (BaseController) para incorporar las funciones más comunes, tales como parámetros de validación. Cada controlador (Page Controller) puede heredar las funcionalidades comunes de la clase base. Además de heredar una clase base común, también se pueden definir un conjunto de clases de ayuda (helpers) que los controladores pueden llamar para realizar funciones habituales. (Trowbridge, Mancini, Quick, Hohpe, Newkirk, & Lavigne, 2012)

- **ASP.NET MVC**

El patrón de arquitectura Modelo-Vista-Control (MVC) separa una aplicación en tres componentes principales: El modelo, la vista y el controlador. El framework ASP.NET MVC provee una alternativa al modelo clásico de desarrollo ASP.NET Web Forms con el objetivo de crear aplicaciones web basadas en MVC. ASP.NET MVC es ligero, altamente orientado al desarrollo de pruebas, se integra con las características que siempre ha tenido ASP.NET. El modelo de desarrollo MVC está definido en la librería System.Web.Mvc y esta a su vez se apoya en la librería System.Web. (ASP.NET, 2012)

ASP.NET MVC es una nueva plataforma para construir aplicaciones ASP.NET, está basada en el mismo ambiente de tiempo de ejecución que las aplicaciones clásicas ASP.NET Web Forms. ASP.NET MVC hace que el desarrollo de aplicaciones web sea una experiencia significativamente diferente al modelo de Web Forms.

ASP.NET MVC fue diseñado para centrarse en las acciones que el usuario puede tomar cuando esta navegando una página. Este tiene un motor de vistas diferente que permite tener mucho más control sobre el código generado. ASP.NET MVC no está basado sobre el patrón Page Controller, se podría decir que opta por una variación orientada a la web del clásico patrón Model-View-Controller (MVC). (Esposito, 2010)

Además posee un conjunto de librerías (ensamblados) que proporcionan las nuevas funcionalidades a nivel de API, incluye plantillas y herramientas que se integran en Visual Studio 2008 y 2010 para facilitarnos un poco las cosas. (Aguilar, 2010)

Las principales características de ASP.NET MVC son:

- Permite una separación clara de las funcionalidades de la aplicación, y el desarrollo guiado por pruebas (TDD- Test Driven Development).

- Es altamente extensible. Todo en el framework MVC está diseñado para que pueda ser personalizado fácilmente (por ejemplo: podemos poner nuestro propio motor de vistas, políticas de ruteo, serialización de parámetros, etc).

- El framework MVC soporta los archivos .ASPX, ASCX y .MASTER como “view templates” (es decir, podemos usar cualquier característica de ASP.NET como las *masterpages* anidadas, <%= %>, *snippets*, controles de servidor declarativos, *templates*, *data-binding*, *localization*, etc). Sin embargo, no usa el modelo de interacción *post-back* para las comunicaciones con el servidor, sino que enruta las interacciones del usuario a una clase controlador.

Soporta todas las características de ASP.NET como las autenticaciones por formularios, por usuario Windows o usando *membership provider, URL authorization, membership/role*, administración del estado de la sesión, monitoreo, sistema se configuración, etc. (Alameda, 2009)

Este modelo de desarrollo esta basado en en patron de diseño MVC y tambien en los patrones Front Controller y Model2 que se explicarán en el desarrollo de este trabajo.

1.3.4 Migración de Aplicaciones

Es un proceso que consiste en hacer que los datos y las aplicaciones existentes funcionen en un computador, software o sistema operativo distinto. En la actualidad este término se ha utilizado mucho, debido al auge del software libre y al hecho de que instituciones públicas a nivel mundial han realizado procesos de migración exitosos. (Wikipedia, 2011)

La migración de datos y sistemas es percibida muchas veces como un aspecto complejo de las aplicaciones. Hay sin embargo numerosos desafíos para una migración de datos exitosa.

Las migraciones implican a menudo altos volúmenes de datos - en algunos casos, se trata de una migración sobre todas las transacciones de la organización. La migración implica el procesamiento de grandes cantidades de datos individuales.

En muchos casos la coherencia necesita ser mantenida entre los viejos y nuevos sistemas, después de que se hayan migrado los datos. Éste es, por ejemplo, el caso cuando aplicaciones múltiples trabajan contra las mismas bases de datos, pero no consiguen ser migradas al mismo tiempo. O cuando un nuevo sistema se pone gradualmente en fase con los usuarios. En estos casos, se puede tener necesidad de una sincronización bidireccional compleja entre los viejos y nuevos sistemas. (Talend, 2010)

2. METODOLOGÍA DEL PROYECTO

Para la construcción de este documento se utilizó como fuente de información principalmente la búsqueda bibliográfica sobre ASP.NET en general y de los dos modelos de desarrollo estudiados; la información fue obtenida básicamente de los sitios que Microsoft provee a los desarrolladores para obtener documentación sobre el tema como lo son MSDN Library y www.asp.net. También fue de gran ayuda el libro Programming Microsoft ASP.NET MVC del autor Dino Esposito, libro en el cual los capítulos iniciales realizan una muy buena descripción de cada uno de los modelos estudiados (ASP.NET Web Forms y ASP.NET MVC) y de los patrones de diseño utilizados y sus diferencias. Esta búsqueda bibliográfica se puede evidenciar en el cumplimiento de los objetivos específicos 1 y 2.

Con base en la información bibliográfica recolectada, los aspectos definidos a estudiar y a opiniones encontradas en algunos blogs y foros de la Internet sobre el tema se procedió a realizar el análisis de ventajas, desventajas; similitudes y diferencias de cada modelo de desarrollo para dar cumplimiento al objetivo específico 3

Para realizar la migración, inicialmente se hizo una búsqueda de las metodologías de migración utilizadas por empresas especializadas en este campo. Posteriormente y con base en un análisis de las características de las metodologías consultadas se planteo una metodología que garantizara el éxito del proyecto.

El ejemplo por utilizar en la ejemplificación del procedimiento es un tutorial para aprender a programar sobre ASP.NET Web Forms que se encontró en la página www.asp.net; inicialmente se construyó el programa como método de aprendizaje de la plataforma utilizando Visual Studio 2010. Se escogió este aplicativo por tratarse de algo simple pero a la vez con las funcionalidades más utilizadas en un desarrollo de este tipo.

Luego con base en la información recolectada y los análisis hechos se paso a realizar la migración. Es importante decir que no se encuentra mucha información sobre el tema por lo que el trabajo se realizó basándose en el análisis de las funcionalidades del aplicativo, en los datos de entrada y salida. Y para el caso de las interfaces de usuario en el código HTML generado. El aplicativo MVC se construyó también sobre la herramienta Visual Studio 2010.

Finalmente aprovechando que el ambiente de ASP.NET MVC facilita el uso de pruebas unitarias se eligió esta herramienta como método de validación de la migración. Las pruebas unitarias permiten validar efectivamente si logró realizar la migración de las funcionalidades de la aplicación, evaluando porciones de código frente a un resultado esperado.

3. PROCEDIMIENTO DE MIGRACIÓN DE ASP.NET WEB FORMS A ASP.NET MVC

3.1 ASP.NET WEB FORMS

3.1.1 Funcionamiento de Web Forms

- **Generalidades**

ASP.NET Web Forms en un modelo de programación enfocado en la productividad y en el desarrollo rápido de aplicaciones (RAD – Rapid Application Development) apoyado en poderosas herramientas con un modelo de programación basado en componentes.

El éxito de ASP.NET Web Forms radica en que abrió el mundo del desarrollo web a miles de desarrolladores con unos muy limitados conocimientos en los lenguajes necesarios para este tipo de proyectos, como lo son HTML, CSS y JavaScript. Esto se logra gracias a que el enfoque de Web Forms es que el desarrollo de aplicaciones para la web sea similar al de aplicaciones para entornos de escritorio. Esto es posible por medio de una serie de controles predefinidos que ayudan a realizar las tareas más comunes como lo son: el acceso a fuentes de datos, la autenticación, presentación de información con ordenamiento y filtros, entre otros. (Esposito, 2010)

Teniendo estos aspectos en cuenta vamos ahora a explicar los pilares en que se fundamente el desarrollo web en Web Forms: los métodos *postback* y el *view state*.

- **Postback**

  Una página en ASP.NET Web Forms está basada en un formulario que contiene todos los elementos para que el usuario pueda interactuar con ellos. El formulario puede contener además otros elementos tales como botones o enlaces. Un *postback* sucede cuando un usuario envía información de un formulario web a una acción en la lógica de negocio (archivo codebehind), este proceso se llama *postback*, porque el envío se hace con payloads usando el método HTTP POST.

  En Web Forms, la página envía el contenido de su formulario a si mismo. En otras palabras, la página es un bloque constituyente de la aplicación que contiene interfaz grafica y algunos procedimientos lógicos para la interacción con el usuario.

  Las similitudes entre una página Web Forms y un formulario Windows son sumamente evidentes, pero en algunos aspectos es mucho menos obvia como en el caso del *postback*. Por ejemplo, si el usuario acciona un botón de la página que es mostrado por un navegador, el clic da la instrucción al navegador de que haga una petición al servidor web de una nueva instancia de la misma página; al hacerlo, el navegador también actualiza el contenido disponible en el formulario de

la página. En el servidor se procesa la petición y se ejecuta el código que se tenga definido para esta.

El siguiente código muestra la relación entre el botón y el código que este ejecuta

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="Button"
OnClick="Button1_Click" />
```

El anterior código, es el fragmento de una página ASPX y que indica que cuando se haga clic se ejecute la función “Button1_Click” que se muestra a continuación.

```
public void Button1_Click(object sender, EventArgs args)
 {
     // Muestra en el label el contenido del textbox
     Label1.Text = "The textbox contains: " + TextBox1.Text;
 }
```

En el momento en que la función “Button1_Click” se ejecuta, todos los controles de servidor de la página (los que están marcados con la etiqueta runat=”server”) son actualizados para mantener exactamente el estado que tenían durante la última solicitud a la página, además de las modificaciones resultantes por la función y finalmente se vuelve a cargar la página actualizada. Para el caso del ejemplo propuesto la función consiste en colocar el texto ingresado en un *textbox* y colocarlo en un *label*. Esto se logra gracias a las propiedades que posee cada uno de los controles.

Este comportamiento del manejo del estado de los controles es de esperarse en ambientes de escritorio, sin embargo en Web Forms esto se logra gracias a la magia de los *postback*. (Esposito, 2010)

- **View State**

El *view state* es un diccionario que las aplicaciones hechas en Web Forms usan para mantener persistente el estado de sus controles cuando realizan dos *postback* consecutivos. El *view state* juega un rol esencial en la implementación del modelo *postback*. El manejo de los estados sería imposible sin el *view state* de Web Forms.

Para resumir: *view state* es el resultado de proceso de ingeniería común con las páginas ASP clásicas. En las páginas ASP clásicas, los desarrolladores frecuentemente ocultaban los campos para hacer seguimiento a los valores críticos después de dos peticiones seguidas. Esto fue necesario cuando existen múltiples formularios HTML en una sola página. Enviando datos de uno solo formulario, los otros limpiaban sus campos, perdiéndose información valiosa. Para compensar este comportamiento, los valores que se deben guardar son almacenados en un campo oculto y empleados para inicializar los campos cuando carga la página.

El *view state* no es más que una extensión y versión mejorada de la solución explicada en el párrafo anterior. El *view state* es un campo oculto único y codificado que guarda un diccionario con los valores de todos los controles de un formulario de una página Web Forms.

Por defecto, cada control de la página guarda su propio estado, con todos los valores de sus propiedades, en el *view state*. En una página promedio, el *view state* ocupa una significativa cantidad de datos extras. Estos datos son descargados al cliente y subidos al servidor por cada petición que haga la página. Sin embargo, estos datos nunca son usados por el cliente.

Por el tamaño que este ocupa y también porque no es muy obvio su rol, el *view state* es considerado con frecuencia como una pesada carga que tiene que llevar cada página de Web Forms o como una forma de perder ancho de banda. Es muy posible escribir páginas que minimizan el uso del *view state* para hacer más ágil el proceso de descarga de archivos en el cliente, pero el *view state* es una pieza fundamental de la arquitectura Web Forms, entonces para eliminar el *view state* Web Forms tendría que ser rediseñado significativamente.

El *view state* tiene para muchas personas una mala reputación, mas como resultado de un mal uso del modelo de desarrollo, muy pocos controles en muy pocos escenarios requieren el uso del *view state* pero muchos se ven atraídos en colocar cosas en el *view state* que no deberían estar allí. El *view state* es de mucho cuidado, un cambio de menor importancia a veces puede tener un campo de afectación bastante grande. (Esposito, 2010)

- **Ciclo de vida de la página**

Al igual que en Windows Forms, hay eventos que se desencadenan en un cierto orden en un Web Forms cuando la página se ejecuta. También hay eventos que se desencadenan en respuesta a la interacción del usuario en el navegador con la página representada. En ocasiones cuando pensamos en cómo un ASP o HTML estándar se crea y se envía a un navegador, se asume que todo se procesa de una manera muy lineal. Sin embargo, para Web Forms, nada podría estar más lejos de la verdad.

Las páginas ASP.NET Web Forms se ejecutan como código de servidor. Por lo tanto, para que la página sea procesada, esta está configurada para enviar la información al servidor cuando los usuarios accionan los botones. Cada vez que sucede esto la página es enviada de nuevo a sí mismo para que pueda ejecutar su código en el servidor de nuevo y luego hacer una nueva versión de sí misma para devolverle al usuario. (MSDN Library, 2012)

El ciclo de procesamiento de una página Web Forms es el siguiente:

1. El usuario solicita la página. (La página se solicita mediante un método HTTP GET.) La página se ejecuta por primera vez.

2. La página es procesada en el servidor y devuelta en código HTML dinámico al navegador, lo que el usuario ve como una página Web similar a cualquier otra página.

   Al momento de ser procesada la página puede pasar por los siguientes eventos, los que el programador puede usar para realizar determinadas funciones, similar a como ocurre en Windows Forms.

   2.1 Page_Init: El servidor crea una instancia del Web Form, carga sus controles y los inicializa con lo recibido del estado de la vista (datos del browser del cliente que vienen con la petición).

   2.2 Page_Load: Los controles del servidor se cargan en un nuevo objeto de tipo Page. Ahora el programador puede acceder a la información de la vista y manejar los controles de la página, contenido, etc.

   2.3 Page_PreRender: El objeto Page va a pasar a código interpretable por el browser en cuanto finalice este evento.

   2.4 Page_Unload: El objeto Page deja de ser referenciable, se destruye.

   2.5 Page_Disposed: La memoria ocupada por Page es liberada.

   2.6 Page_Error: Evento iniciado cuando se ha producido una excepción no capturada.

   2.7 Page_AbortTransaction: Una operación transaccional ha sido finalizada sin llegar a su fin.

   2.8 Page_CommitTransaction: La transacción se ha completado con éxito.

   2.9 Page_DataBinding: Un control de servidor se ha conectado a una fuente de datos. (Garcia Puebla, 2008)

3. El usuario completa la información del formulario o selecciona de las opciones disponibles y luego activa un botón de enviar. (Si el usuario activa un enlace en lugar de un botón, la página podría simplemente navegar a otra página o también ejecutar un postback)

4. La página se envía al servidor Web. (El navegador realiza un método HTTP POST, que en ASP.NET se conoce como una devolución de datos.) En concreto, la página se devuelve a sí misma. Por ejemplo, si el usuario está trabajando con la página Default.aspx, al activar un botón de la página, la página va al servidor con la indicación de que debe volver a Default.aspx

5. En el servidor Web, la página se ejecuta de nuevo. La información que el usuario ha introducido o seleccionado está disponible en la página.

6. La página realiza el procesamiento que se haya programado.

7. La página vuelve a presentarse actualizada en el navegador.

Este ciclo continúa mientras el usuario está trabajando en la página. Cada vez que el usuario activa un botón, la información en la página se envía al servidor Web y la página se ejecuta de nuevo. Cada ciclo se conoce como una ida y vuelta. Dado que el procesamiento de página se produce en el servidor Web, cada acción que puede hacer la página requiere un viaje de ida y vuelta al servidor.

- **Patrón Page Controller y Web Forms**

El patrón Page Controller se utiliza para aceptar la entrada de las solicitudes a la página, invocar las solicitudes requeridas en el modelo, y determinar la vista correcta a utilizar para presentar los resultados. Siempre separando la lógica de negocio del código de las vistas. Dependiendo del caso, creando una clase base común para todos los controladores de la página para evitar la duplicación de código y una mayor coherencia y capacidad de prueba. La figura 3 muestra cómo el controlador de la página se relaciona con el modelo y la vista.

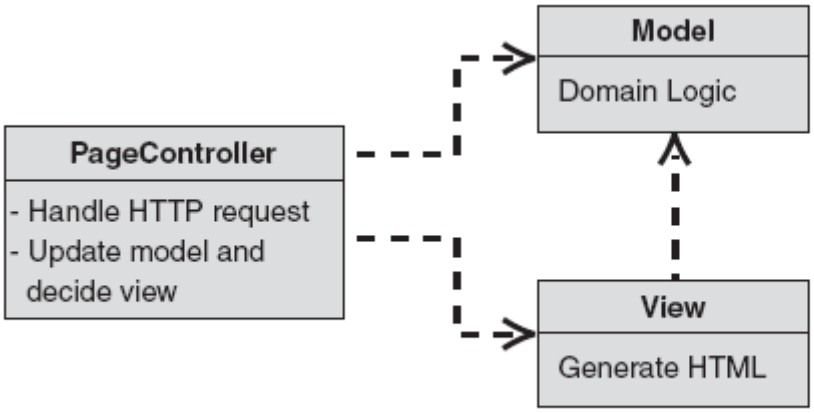

Figura 3 - Diagrama Page Controller

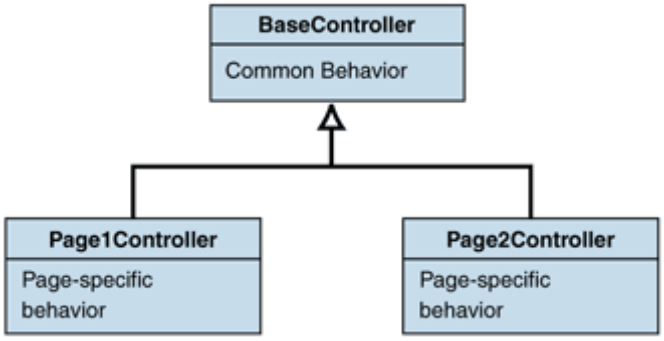

Figura 4 – Base Controller

Los conceptos descritos en el patrón Page Controller están implementados por defecto en ASP.NET Web Forms. Una página en Web Forms implementa estos conceptos en la manera como la página captura un evento del cliente, transmite este al servidor y llama al método apropiado; este proceso es automático y no es muy visible la forma en que este se implementa. El controlador de la página es extensible en donde se exponen varios eventos en puntos específicos del ciclo de vida de la página, entonces la aplicación y sus acciones correrán cuando sea el momento apropiado.

Por ejemplo, si se asume que el usuario que interactúa con una página Web Forms que contiene el control de servidor que dibuja un botón (button server control). Cuando el usuario active ese botón un evento será transmitido como un HTTP *post* al servidor, donde ASP.NET *page framework* interpreta la información enviada y la asocia el evento con el controlador de eventos adecuado. El *framework* automáticamente llama el controlador de eventos apropiado para el botón como parte normal del procesamiento del *framework*, como resultado no se necesitará implementar esta funcionalidad. Se puede usar el controlador incorporado (el descrito anteriormente y que viene por defecto) o también se puede remplazar el controlador incorporado con un controlador personalizado por el usuario.

El uso patrón Page Controller nos presenta las siguientes ventajas:

- Simplicidad
- Esta incorporado en las características del Framework ASP.NET Web Forms
- Reutilización de componentes
- Capacidad de expansión
- Separación de responsabilidades en equipos de desarrollo

Y a la vez posee las siguientes desventajas:

- Un controlador por página

- Extensos árboles de herencia
- Dependencia del marco de desarrollo

ASP.NET Web Forms implementado de la forma correcta es una excelente opción para el desarrollo de aplicaciones en la web, lo malo es que muchos desarrolladores debido a la libertad que proporciona el modelo de desarrollo, utilizan esta herramienta de una manera incorrecta haciendo uso, no de un patrón, sino de un anti-patrón que es llamado Smart UI (Malcolm, 2012); este consiste en una aplicación donde no hay una clara separación de conceptos, es decir, donde interfaz grafica (para nuestro caso HTML y controles de servidor) y la lógica de negocio o lógica de la aplicación se hallan mezclados, lo que hace que la aplicación se vuelva desordenada y muy difícil de realizarle mantenimiento, de modificar y de escalar.

3.1.2 Acceso a bases de datos

Cualquier aplicación web frecuentemente necesita acceder a fuentes de datos para el almacenamiento y consulta de los mismos. Se puede escribir código para acceder a los datos utilizando las clases del espacio de nombres System.Data (comúnmente conocido como ADO.NET) y desde el espacio de nombres System.Xml. Este enfoque es común en las versiones anteriores de ASP.NET.

Sin embargo, ASP.NET también permite realizar enlaces de datos declarativos. Para ello es necesario ningún código en absoluto para los escenarios de datos más comunes, incluyendo:

- Selección y visualización de datos.
- Ordenación, paginación y almacenamiento en *caché* de datos.
- Actualizar, insertar y eliminar datos.
- Filtrado de datos utilizando los parámetros de tiempo de ejecución.
- Creación de escenarios maestro-detalle utilizando parámetros.

ASP.NET, en este caso tanto Web Forms como MVC, incluye varios tipos de controles de servidor que participan en el modelo de datos declarativo, incluidos los controles de origen de datos, los controles enlazados a datos, y el control extensor de consulta. Estos controles de gestión de las tareas fundamentales que son requeridos por el modelo de la Web sin estado para mostrar y actualizar datos en páginas Web ASP.NET. Los controles permiten añadir comportamiento de enlace de datos a una página sin tener que entender los detalles del ciclo de vida de la página.

Los controles de origen de datos son controles ASP.NET que administran las tareas de conexión a un origen de datos y además de eso la lectura y escritura de datos. Dichos controles no tienen ninguna interfaz de usuario, sino que actúan como intermediarios

entre un almacén de datos en particular (como una base de datos, objetos de empresa, o un archivo XML) y otros controles de la página Web ASP.NET. (MSDN Library, 2012)

ASP.NET incluye los controles de origen de datos siguientes:

- AccessDataSource: Permite utilizar una base de datos Microsoft Access

- LinqDataSource: Le permite utilizar Language-Integrated Query (LINQ) en una página Web ASP.NET mediante marcado declarativo para consultar y modificar datos de un objeto de datos. Admite la generación automática de consultas, actualizaciones, inserciones y eliminaciones. El control también admite la ordenación, el filtrado y la paginación.

- ObjectDataSource: Permite trabajar con un objeto de negocio u otras clases de objetos y crear aplicaciones web basadas en objetos de nivel medio para administrar los datos.

- SqlDataSource: Permite trabajar con ADO.NET gestionado proveedores de datos, que proporcionan acceso a Microsoft SQL Server, OLE DB, ODBC o bases de datos de Oracle.

- EntityDataSource: Le permite al desarrollador enlazar datos, se basa en el Entity Data Model (EDM). Admite la generación automática de comandos de actualización, inserción, eliminación y consulta. El control también admite la ordenación, filtrado y paginación.

- XmlDataSource: Permite trabajar con un archivo XML, especialmente para los controles de servidor ASP.NET jerárquicos como TreeView o el control Menu. Soporta funciones de filtrado mediante expresiones XPath y le permite aplicar una transformación XSLT a los datos. El XmlDataSource permite actualizar los datos al guardar el documento XML completo con los cambios. (MSDN Library, 2012)

Además en la parte de la presentación de datos, existen unos controles llamados controles de enlace de datos, aunque son controles de servidor, tema que se tratará en la sección 3.1.3; es importante hablar de ellos en este punto ya que su funcionalidad involucara tanto el acceso a datos como la presentación. Un control de enlace de datos se conecta con un control de origen de datos y captura los datos automáticamente en el momento apropiado del ciclo de vida de la página. Dichos controles pueden tomar ventaja de las capacidades proporcionadas por un control de origen de datos, incluida la clasificación, búsqueda, almacenamiento en caché, filtrado, actualización, eliminación e inserción. Los principales controles de enlace de datos son:

- AdRotator: Permite visualizar anuncios en una página como una imagen a la que los usuarios pueden hacer clic para ir la URL del anuncio.

- DataList: Muestra datos en una tabla.

- DetailsView: Muestra un registro a la vez en un diseño tabular y permite editar, eliminar e insertar registros.

- FormView: Al igual que el control DetailsView, pero permite definir un diseño de forma libre para cada registro. El control FormView es como un control DataList para un único registro.

- GridView: Muestra los datos en una tabla e incluye compatibilidad para editar, actualizar, eliminar, ordenar y paginar datos sin necesidad de código.

- ListView: Permite definir la disposición de los datos mediante el uso de plantillas. Soporta edición, inserción y eliminación

- Menu: Presenta los datos en un menú jerárquico dinámico que puede incluir submenús.

- Repeater: Representa los datos en una lista. Cada elemento se representa utilizando una plantilla de elementos que se defina.

- TreeView: Muestra los datos en un árbol jerárquico de nodos expandibles. (MSDN Library, 2012)

3.1.3 Presentación de datos

- **Navegación**

En ASP.NET Web Forms las direcciones de los sitios se dan por enrutamiento como lo hace el modelo MVC donde las URL de la página son más lógicas que físicas, en Web Forms las URL se refieren normalmente a un archivo físico de extensión aspx, es decir el archivo que contiene el formulario y que a su vez referencia un archivo de control (*codebehind*) que controla las acciones que se deben hacer cuando el formulario es mandado al servidor. Se tiene que por cada archivo aspx, hay un archivo controlador *codebehind*, es aquí donde se observa el funcionamiento del patrón Page Controller.

Los archivos aspx se pueden encontrar en la raíz del proyecto o bien dentro de una subcarpeta la que se debe referenciar también dentro de la URL. El archivo aspx también puede llevar parámetros (uno o muchos), que le ayudan a la aplicación a mostrar un contenido especifico. Por ejemplo, una solicitud de http://server/application/Products.aspx?id=4 se asigna a un archivo que se denomina Products.aspx que contiene código y marcado para representar una respuesta al navegador. La página Web utiliza el valor de cadena de consulta de id = 4 para determinar qué tipo de contenido a visualizar. (MSDN Library, 2012)

Resumiendo estos son los tipos de URL que podemos encontrar en Web Forms

- Archivo en la raíz del proyecto sin parámetros:

  http:// [server] / [aplicación] / [archivo].aspx

- Archivo en una subcarpeta

http:// [server] / [aplicación] / [carpeta] / [archivo].aspx

- Archivo con parámetros
- http:// [server] / [aplicación] / [carpeta] / [archivo].aspx?[parámetro=value]

○ **Master Pages**

Las *master pages* o páginas maestras ASP.NET le permiten crear un diseño coherente para las páginas de una aplicación web. Una sola página maestra puede definir el aspecto y comportamiento estándar que desea para todas las páginas (o un grupo de páginas) de la aplicación. Una master page es una estructura base para un conjunto de páginas pertenecientes a un mismo sitio Web. Este esqueleto base se almacena en un archivo independiente y luego es heredado por otras páginas que requieren esa estructura base. (MSDN Library, 2012)

Un ejemplo donde utilizar una página Master Page es la definición de la cabecera y el pie del sitio, luego todas las otras páginas del sitio heredan de dicha página. La ventaja fundamental en este ejemplo es que si hay cambios en la cabecera del sitio solo debemos hacerlos en la página principal y con esto se propaga en todas las páginas que heredan de la misma.

Para el contenido que es dinámico cambia de página a página se utilizan unos controles llamados ContentPlaceHolder, se pueden definir tantos como sean necesarios y ya en cada página que se tenga en la aplicación se agrega el contenido y al ejecutarse la aplicación el servidor enviará al cliente una mezcla entre la *master page* y el contenido de los ContentPlaceHolder. Esto es de gran ayuda y evita la duplicación de código. En la figura 5 se ejemplifica más claramente el funcionamiento de la *master page*.

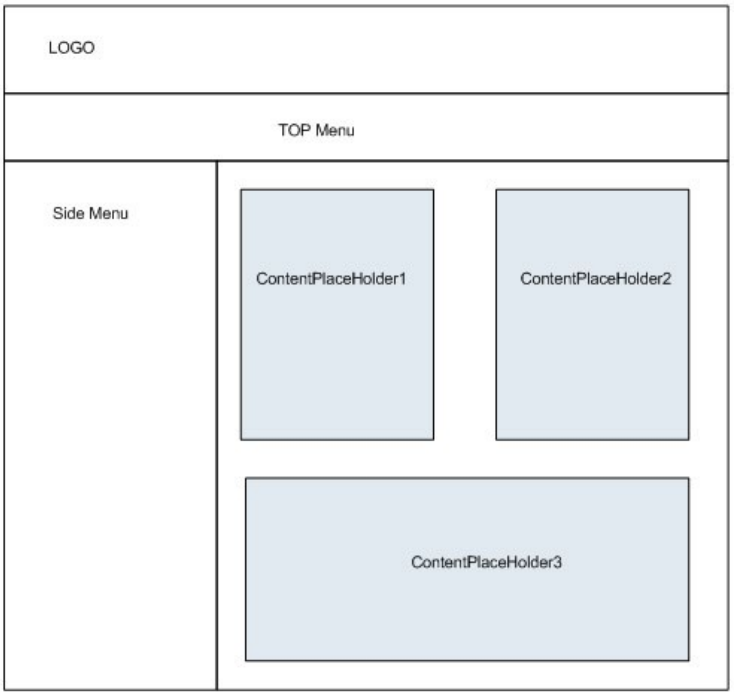

Figura 5 - Master Pages

- **Controles de servidor**

Los controles de servidor son parte fundamental del modelo Web Forms. La salida de una página ASP.NET Web Forms hacia el navegador que es código HTML, se define en el servidor mediante una combinación de literales HTML y formato de los controles de servidor ASP.NET. Un control de servidor es un componente con una interfaz pública que se puede configurar con etiquetas de marcas, etiquetas y atributos. Cada control de servidor se caracteriza el atributo runat=server y por una ID única que la define completamente.

En las páginas Web Forms la diferencia entre un control de servidor y el código HTML es la presencia del atributo *runat*. Cualquier cosa que carezca del atributo *runat* se trata como HTML literal y se emite a la secuencia de respuesta de salida. Una instancia de control de servidor es procesada y emite un código HTML de salida según los atributos que posea. (Esposito, 2010)

Los controles de servidor generan automáticamente código HTML y JavaScript. La programación de un control de servidor es tan fácil como configurar las propiedades de un componente de Windows Forms, cada uno posee sus atributos y funcionalidades especiales es por esto que no es muy necesario tener un gran conocimiento de HTML para programar en Web Forms y por lo que muchas veces el código generado no es compatible con los estándares de todos los navegadores, lo que produce problemas de visualización.

Existe una gran variedad de controles de servidor de Web Forms, la página MSDN Library los clasifica de la siguiente manera: (MSDN Library, 2012)

1. **Controles de servidor HTML:**

   Controles de servidor HTML son elementos HTML que contienen atributos que los hacen programable en código de servidor. De forma predeterminada, los elementos HTML en una página Web Forms no están disponibles en el servidor. En su lugar, se tratan como texto que pasa directo al navegador. Sin embargo, mediante la conversión de los elementos HTML a los controles de servidor HTML, es posible exponerlos como elementos que se pueden programar en el servidor.

   Un ejemplo de un control de servidor HTML es el siguiente:

```
<input id="Name" type="text" size="40" runat="server" />
```

2. **Controles de servidor Web:**

   Son los controles que más características incorporadas posee. Los controles de servidor Web incluyen no sólo los controles de formulario, como botones y cuadros de texto, sino también otros para fines especiales tales como un calendario, menús, y un control de vista de árbol (*tree view control*). Estos controles son más abstractos que los controles de servidor HTML, ya que no reflejan necesariamente la sintaxis HTML.

   Un ejemplo de este tipo de control es el siguiente:

```
            <asp:Label ID="LabelCartHeader" runat="server"
Text="Please check all the information below to be sure it&#39;s correct.">
        </asp:Label>
```

   Este control cuando es procesado por el servidor se convierte en la etiqueta HTML Label.

3. **Controles de Validación:**

   Estos controles incorporan una lógica que le permite al programador validar controles de entrada tales como el control TextBox. Los controles de validación le permiten comprobar si un campo obligatorio, verificar si cumplen con un valor específico o un patrón de caracteres, para verificar que un valor se encuentra dentro de un rango, etc.

4. **Controles de usuario:**

   Estos controles se crean como otras páginas Web ASP.NET. Puede incrustar (embeber) controles de usuario ASP.NET en otras páginas Web ASP.NET, esta

que es una manera fácil de crear barras de herramientas y otros elementos reutilizables.

3.1.4 Seguridad

La seguridad en ASP.NET, tanto en Web Forms como en MVC consiste básicamente en los procedimientos de autenticación y autorización. La autenticación es un proceso que consiste en obtener las credenciales de identificación, como nombre y contraseña, de un usuario y validarlas consultando a una autoridad determinada. Si las credenciales son válidas, se considera a la entidad que ha enviado las credenciales como una entidad autenticada. Una vez autenticada la entidad, el proceso de autorización determina si esa entidad tiene acceso a un recurso específico. La autorización es el procedimiento que le indica al usuario si puede o no acceder a un recurso determinado.

La autenticación utilizada por ASP.NET se logra a través de la colaboración del IIS para a través de la consola de administración de IIS usted puede definir que tipo de autenticación que va utilizar, siempre en concordancia con lo definido en el archivo de configuración del proyecto.

Hay cuatro tipos opciones de autenticación de Windows disponibles que se pueden configurar en IIS:

- Anonymous Authentication: IIS no realiza ninguna comprobación de autenticación. IIS permite a cualquier usuario acceder a la ASP. NET.

- Autenticación básica: Para este tipo de autenticación, un nombre de usuario y contraseña de Windows tienen que ser proporcionados a conectar. Sin embargo, esta información se envía a través de la red en texto plano y por lo tanto, éste es un método inseguro de autenticación.

- Autenticación implícita: Es igual que la autenticación básica, pero la contraseña es encriptada antes de ser enviada a través de la red. Sin embargo, para utilizar la autenticación implícita, se debe usar IE 5.0 o superior.

- La autenticación integrada de Windows: En este tipo de técnica de autenticación en que las contraseñas no se envían directamente a través de la red. La aplicación utiliza tanto el protocolo kerberos o protocolos de challenge/response para autenticar a los usuarios. Kerberos es un protocolo de autenticación de red, se ha diseñado para proporcionar una autenticación segura para las aplicaciones cliente-servidor. Proporciona las herramientas de autenticación y criptografía fuerte sobre la red para garantizar que la información fluya de manera segura. (Venulopagan, 2012)

Es necesario entonces que lo que este configurado en el IIS sea concordante con lo definido en el proyecto específicamente en el archivo de configuración (Web.config). En ASP.NET, independiente del modulo que se desarrolle sea ASP.NET Web Forms o ASP.NET MVC, existen tres métodos de autenticación: Forms Authentication, Windows

Authentication y Passport Authetication, los que se detallarán mas adelante. Antes de eso es necesario describir ASP.NET Membership un mecanismo que permite construir complejos sistemas de validación de usuarios.

○ Membership Provider

Membership Provider proporciona una forma integrada de validar y almacenar la información de los usuarios. Por lo tanto ASP.NET Membership gestiona la autenticación y autorización de los usuarios. Membership Provider puede utilizar la autenticación por formularios usando los controles para autenticación de ASP.NET, todo en conjunto permite construir un complejo y seguro sistema para la autenticación.

ASP.NET Membership Provider es útil para:

- Creación de nuevos usuarios y contraseñas
- Almacenamiento de la información de los usuarios (nombres de usuario, contraseñas y datos de apoyo) en Microsoft SQL Server, Active Directory, o un almacén de datos alternativo.
- Autenticar a los usuarios que visitan un sitio web. Puede autenticar los usuarios mediante programación, o puede utilizar los controles ASP.NET de inicio de sesión para crear un sistema de autenticación completa que requiere poco o ningún código.
- Gestión de contraseñas, que incluye la creación, modificación y restablecimiento. Dependiendo de las opciones que usted elija, Membership Provider también puede proporcionar un sistema automatizado para restablecer la contraseña del sistema por medio de una pregunta suministrada por el usuario y su respectiva la respuesta.
- El uso de una identificación única para los usuarios, que se pueden utilizar en otras aplicaciones.
- El usuario tiene la posibilidad de especificar un Membership Provider personalizado, que le permite sustituir su propio código para administrar la pertenencia y mantener los datos de afiliación en un almacén de datos personalizado.

○ Forms Authentication

Forms Authentication o Autenticación por formularios es un sistema mediante el cual las solicitudes que no están autenticadas, son re direccionadas a un formulario HTML. El usuario ingresa y envía sus credenciales. Si la aplicación autentica la solicitud, esta emite una etiqueta de autenticación que se añade a la *cookie* de la sesión del usuario.

Generalmente la base de datos de usuario se encuentra en el Membership Provider, aunque también se puede utilizar una base de datos propia o personalizar el Membership Provider.

- **Windows Authentication**

Windows Authentication o autenticación Windows autentica a los usuarios sobre la base de cuentas de los usuarios de Windows (Directorio activo de usuarios del dominio para el caso de una red corporativa). La autenticación de Windows en ASP.NET en realidad se basa en IIS para hacer la autenticación. IIS se puede configurar para que sólo los usuarios en un dominio de Windows puedan entrar, si un usuario intenta acceder a una página y no está autenticado, se le muestra un cuadro de diálogo pidiéndole que introduzca su nombre de usuario y contraseña. Esta información se pasa al servidor web y se comprueba con la lista de usuarios en el dominio. Si el usuario ha suministrado las credenciales válidas, se concede el acceso. Finalmente el identificador del usuario pasa al motor ASP.NET.

- **Passport Authentication**

Autenticación Passport, también llamado Live es un servicio de autenticación centralizado. Este utiliza el servicio de Microsoft Passport para autenticar a los usuarios de una aplicación. Es necesario que el modo de autenticación de la aplicación este configurado en Passport y que los usuarios se hayan suscrito a un servicio de Passports de Microsoft, por ejemplo Hotmail.

Passport Authentication utiliza un mecanismo de la *cookie* cifrada para identificar e indicar los usuarios autenticados. Si los usuarios ya han visitado la página antes, cuando visite la página de la aplicación de nuevo, ASP.NET lo considera como autenticado. De lo contrario, los usuarios serán redirigidos a los servidores Passport para poder ingresar. Al iniciar sesión, el usuario es redirigido a la página Web ASP.NET que inicialmente trató de acceder.

3.2 ASP.NET MVC

3.2.1 Funcionamiento de MVC

- **Generalidades**

ASP.NET MVC es una nueva plataforma para construir aplicaciones ASP.NET. Basado en el mismo entorno de ejecución de ASP.NET Web Forms, ASP.NET MVC hace que el desarrollo de aplicaciones web sea una experiencia muy diferente que el modelo de Web Forms.

ASP.NET MVC fue diseñado para centrarse en las acciones que un usuario puede tomar cuando se navega por una página. Tiene un motor de vista diferente y permite un control mucho mayor sobre el marcado generado. ASP.NET MVC no tiene en cuenta el modelo de controlador de la página y opta por un modelo diferente que puede ser considerado una variación orientada a la web del clásico Modelo-Vista-Controlador (MVC).

Cuando se desarrolla una aplicación ASP.NET MVC, se piensa en términos de controladores y acciones. El problema que se debe enfrentar en este tipo de desarrollos es como exponer los controladores a los usuarios y la forma de pasar los datos a las vistas desde los controladores. Cada solicitud se resuelve mediante la invocación de un método en una clase de controlador. Ningún *postback* es necesario para resolver una petición, ni tampoco existe el *view state* para conservar el estado de la página. Por último, no existen controles de servidor HTML. Todo esto le quita una gran carga al servidor y hace de las aplicaciones MVC, aplicaciones mucho más livianas que las aplicaciones Web Forms.

Sin embargo, si nos vamos a un nivel mucho más bajo a ASP.NET MVC, está claro que su forma de trabajar se basa todavía en el manejo de solicitudes HTTP, excepto que la dirección URL se trata de una manera diferente y para cualquier acción resultante de una solicitud se utilizan otras acciones del controlador en lugar de *postbacks*. En general, el modelo de programación de ASP.NET MVC, presenta un nuevo paradigma y plantea nuevos desafíos a los desarrolladores. (Esposito, 2010)

A continuación se describen las principales ventajas de MVC:

- Hace que sea más fácil de manejar la complejidad al dividir una aplicación en el modelo, la vista y el controlador.

- No utiliza el *view state* ni tampoco controles de servidor.. Esto hace ideal al *framework* MVC para los desarrolladores que desean tener un control total sobre el código HTML generado por la aplicación web

- Se utiliza el patrón Front Controller para procesar las peticiones a la aplicación a través de un único controlador. Esto permite diseñar una aplicación que soporta una buena infraestructura de enrutamiento dinámico, con URLs mucho más limpias y entendibles.

- Se ofrece un mejor soporte para desarrollo basado en pruebas (TDD)

- Funciona bien para las aplicaciones Web que se apoyan en grandes equipos de desarrolladores y diseñadores web que necesitan un alto grado de control sobre el comportamiento de la aplicación. (MSDN Library, 2012)

○ Estructura de un proyecto MVC

Un proyecto ASP.NET MVC presenta por defecto la estructura de directorios que se muestra en la siguiente figura y que se explica a continuación:

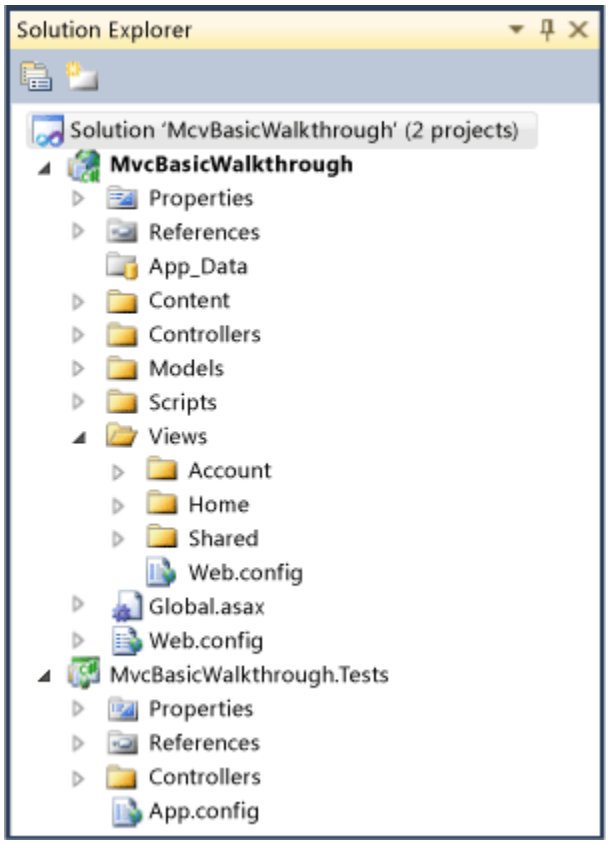

Figura 6 - Estructura de un proyecto MVC

- Content: es la ubicación recomendada para agregar archivos de contenido, como las hojas de estilo CSS, imágenes, etc. En general, la carpeta de contenido es para los archivos estáticos.

- Controllers: es la ubicación recomendada para los controladores. El marco MVC requiere que los nombres de todos los controladores para terminar con "Controller", como HomeController, LoginController o ProductController.

- Models: aquí se almacenan las clases que representan el modelo para la aplicación web MVC. Esta carpeta normalmente incluye código que define los objetos y que define la lógica para la interacción con una fuente de datos. Típicamente, los objetos reales del modelo será en bibliotecas de clases separadas. Sin embargo, cuando se crea una nueva aplicación, es posible colocar las clases aquí y luego trasladarlas a las bibliotecas de clases separadas en un momento posterior en el ciclo de desarrollo.

- Scripts: es la ubicación recomendada para los archivos de secuencia de comandos (scripts) que soporta la aplicación. De forma predeterminada, esta carpeta contiene archivos de ASP.NET AJAX *foundation* y la biblioteca jQuery.

- Views: es la ubicación recomendada donde se guardan todas las vistas. Dichas vistas pueden ser ViewPage (. aspx), ViewUserControl (. ascx) y archivos maestros (ViewMasterPage). La carpeta Vistas contiene una carpeta para cada controlador, la carpeta se nombran con el prefijo controller-name. Por ejemplo, si se tiene un controlador llamado HomeController, la carpeta Vistas contendrá una carpeta llamada Home. Por defecto, también hay una carpeta con el nombre compartido (*shared*) de la carpeta Views, que no se corresponde con ningún controlador. La carpeta compartida se utiliza para las vistas que se comparten entre varios controladores. Por ejemplo, usted puede colocar la página maestra (master page) en la carpeta compartida. (MSDN Library, 2012)

Además dentro de la estructura del proyecto existen dos archivos de suma importancia: el Global.asax y el Web.config. El global.asax es donde se encuentran configuradas las rutas que va aceptar la aplicación, por defecto viene con las estructura [Controller]/[View]/[id] pero se puede cambiar y redefinir. En el web.config está definida toda la configuración del proyecto, este es un archivo XML que controla la carga de módulos, configuraciones de seguridad, cadenas de conexión a bases de datos, configuraciones del estado de la sesión, opciones de compilación y el lenguaje de la aplicación. Este archivo de configuración se encuentra también en los proyectos ASP.NET Web Forms y cumple exactamente la misma función.(MSDN Library, 2012)

ASP.NET MVC no obliga a usar siempre esta estructura, las plantillas de proyecto por defecto utilizan este modelo y se recomienda como una manera fácil de estructurar la aplicación. A menos que se tenga una buena razón para utilizar un archivo de diseño alternativo, se recomienda usar este patrón por defecto.

- **Ciclo de vida de la página**

Las solicitudes (*requests*) a una aplicación ASP.NET MVC primero pasan a través del objeto UrlRoutingModule, que es un módulo HTTP. Este módulo analiza la solicitud y realiza la selección de ruta. El objeto UrlRoutingModule selecciona la primera ruta que coincida con la solicitud actual. Si no coinciden las rutas, el objeto UrlRoutingModule no hace nada y deja que la solicitud de recurrir a la habitual ASP.NET o se sigue el procedimiento definido para solicitudes fallidas del IIS. (MSDN Library, 2012)

La siguiente tabla muestra claramente el procedimiento que sigue la ejecución de una aplicación MVC:

Tabla 1 - Proceso de ejecución de una aplicación MVC

Etapa	**Detalle**
La aplicación recibe la solicitud o petición (request) por parte del cliente	En el archivo Global.asax, la ruta se añade al objeto RouteTable.
Se realiza enrutamiento	El módulo UrlRoutingModule utiliza el primer objeto que

	coincida en la colección RouteTable con la ruta dada, para crear el objeto RouteData, que después utiliza para crear un objeto RequestContext.
Crear el controlador MVC de la solicitud (request handler)	El objeto MvcRouteHandler crea una instancia de la clase MvcHandler y pasa la instancia RequestContext al controlador.
Creación del controlador	El objeto MvcHandler utiliza una instancia de RequestContext para identificar el objeto IControllerFactory (normalmente una instancia de la clase DefaultControllerFactory) para crear con esta la instancia de controlador.
Ejecución del controlador	La instancia MvcHandler ejecuta al controlador.
Invocación de la acción	El objeto ControllerActionInvoker que está asociado con el controlador determina qué acción de la clase del controlador va llamar, y la ejecuta.
Resultado de la ejecución	La acción recibe la entrada del usuario, prepara los datos de respuesta apropiados, luego la ejecuta e incorpora los datos a retornar en alguno de los siguientes tipos de resultados: ViewResult (que representa una vista y es el tipo de resultado más frecuentemente utilizado), RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult, y EmptyResult.

- **ASP.NET MVC y los patrones de diseño**

Patrón MVC (Modelo-Vista-Controlador)

La arquitectura Modelo Vista Controlador es un patrón de diseño y también una arquitectura que consiste en la separación de estos tres conceptos, dividiendo funcionalmente la aplicación de la siguiente manera:

- Modelo:

  El modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El modelo maneja el comportamiento y los datos del dominio de aplicación; responde a las solicitudes de información acerca de su estado (por lo general a la vista), y responde a las peticiones para cambiar el estado (generalmente hechas por el controlador) (Trowbridge, Mancini, Quick, Hohpe, Newkirk, & Lavigne, 2003). El modelo no tiene conocimiento específico de los controladores o de las vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuando cambia el modelo.

- Vista

  La vista es el objeto que maneja la presentación visual de los datos representados por el modelo. Genera una representación visual del modelo y muestra los datos al usuario. Interactúa con el modelo a través de una referencia al propio modelo.

- Controlador

  El controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el modelo a través de una referencia al propio modelo. (Cordova Díaz, 2010)

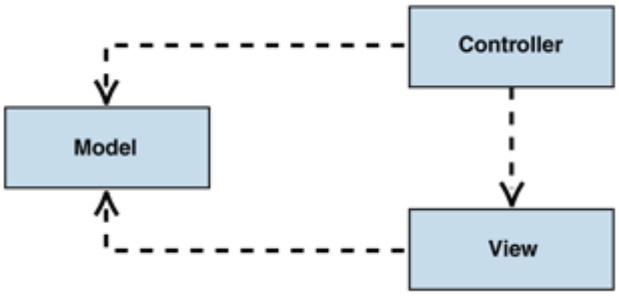

Figura 7 - Estructura MVC

Es importante señalar que tanto la vista como el controlador dependen del modelo. Sin embargo, el modelo no depende de la vista ni del controlador. Esta es una de las principales ventajas de la separación. Esta separación permite que el modelo que se construyó y probó independiente de la presentación visual. Por otro lado, la separación entre la vista (el navegador) y el controlador (los componentes del lado del servidor que maneja la petición HTTP) está muy bien definida.

Modelo-Vista-Controlador es un patrón de diseño fundamental de la separación de la lógica de la interfaz de usuario de la lógica de negocio. Desafortunadamente, la popularidad del patrón ha producido en una serie de descripciones erróneas. En particular, el término "control" se ha utilizado para referirse a cosas diferentes en contextos diferentes. Afortunadamente, la llegada de las aplicaciones Web ha ayudado a resolver algunas de las ambigüedades, porque la separación entre la vista y el controlador es tan evidente.

Este patrón MVC posee las siguientes ventajas:

- Clara separación de responsabilidades entre interfaz, lógica de negocio y de control, que además provoca parte de las ventajas siguientes.

- Simplicidad en el desarrollo y mantenimiento de los sistemas.

- Reutilización de los componentes.
- Facilidad para desarrollar prototipos rápidos.
- Sencillez para crear distintas representaciones de los mismos datos.
- Los sistemas son muy eficientes, ya que ofrecen una mejor respuesta y un menor uso de los recursos. Y a la postre MVC resulta más escalable, es decir facilita la adición de nuevas funcionalidades.

Y sus limitaciones son:

- La complejidad
- Tener que ceñirse a una estructura predefinida, lo que a veces puede incrementar la complejidad del sistema. Hay problemas que son más difíciles de resolver respetando el patrón MVC.

Patrón Front Controller

Cuando se decide usar el patrón MVC para separar la interfaz de usuario de la lógica de negocio en una aplicación web, y se ha revisado el patrón Page Controller pero este patrón hace que la lógica se haga complicada debido a la utilización de jerarquías de herencia demasiado complejas o la aplicación necesita de una navegación entre páginas este basada en una serie de reglas configurables. Se necesita entonces una manera de estructurar el controlador para aplicaciones web complejas que permitan flexibilidad pero eviten la duplicación de código, para esto se utiliza Front Controller. (Trowbridge, Mancini, Quick, Hohpe, Newkirk, & Lavigne, 2012)

El patrón Front Controller implica el uso de un componente centralizado que maneja todas las peticiones entrantes y las envía a otro componente para que este realice las acciones pertinentes y así completar el flujo de la petición. Pero a diferencia del patrón Page Controller donde hay un controlador (handler) diferente para cada petición y este quien determina a que URL debe ir la petición, en Front Controller todas las peticiones son manejadas por un solo componente que es llamado MVC HTTP handler. Esta clase común contiene la lógica que analiza la URL y decide qué controlador se debe a atender la solicitud y que componente de vista se debe a generar el código HTML resultante. El controlador es una clase normal con métodos públicos, y cada método se ejecuta una acción siguiendo los parámetros del usuario. En la figura 8, se puede apreciar las diferencias entre Page Controller y Front Controller. (Esposito, 2010)

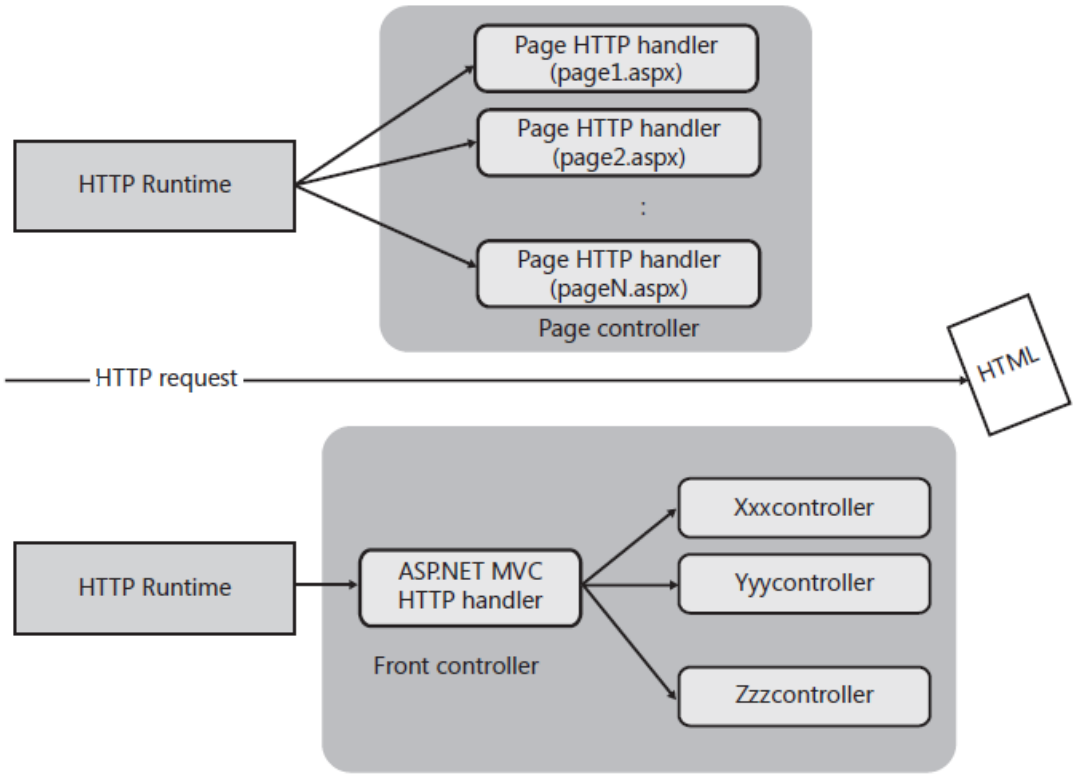

Figura 8 - Page Controller vs Front Controller

Front Controller es un patrón basado en el patrón MVC pero con muchas más mejoras, las siguientes características de MVC aplican también para Front Controller:

- Si la lógica esta repetida en varias vistas en el sistema se necesitará centralizar esto para evitar la duplicación de código, lo que evita problemas a la hora de realizar mantenimientos a las aplicaciones.

- La consulta de datos es mejor manejada si se hace desde un solo lugar que si se tiene una conexión a la base de datos por cada vista que presenta información. Las interfaces gráficas donde los mismos datos en diferentes formas en un buen ejemplo.

- Realizar pruebas sobre las interfaces de usuario es muy complicado y tedioso, separar la aplicación en partes más pequeñas hace que sea más fácil de probar, como lo sugiere el patrón MVC.

MVC, Front Controller y Model2

El patrón MVC se diseñó orientado a las aplicaciones de escritorio, pero debido a su formulación relativamente flexible, se adapta fácilmente para la Web. Una variación web popular del patrón MVC es Model2. Model2 es el nombre del patrón que alimenta a ASP.NET MVC.

El patrón Model2 expone la vista a los usuarios a través de un navegador. Las acciones del usuario son capturados por el navegador y se transforman en una petición HTTP. De esta manera, las solicitudes van desde el navegador a un componente especial (Front Controller) que hace las veces de servidor web. Front Controller se encarga de coordinar todas las peticiones que se hacen a la aplicación web.

En ASP.NET, Front Controller hace el enrutamiento de URL HTTP. El módulo HTTP captura la solicitud y la envía al controlador adecuado para la acción solicitada. Como la acción retorna datos, el controlador le da la orden de pasar los datos nuevos a la interfaz de usuario. La salida de la vista es capturada por el modulo HTTP de Front Controller y enviada al navegador. (Esposito, Cutting Edge: ASP.NET Presentation Patterns - MSDN Magazine, 2008)

La siguiente figura muestra el diagrama de secuencia para una solicitud de acuerdo con Front Controller + Model2. (Esposito, 2010)

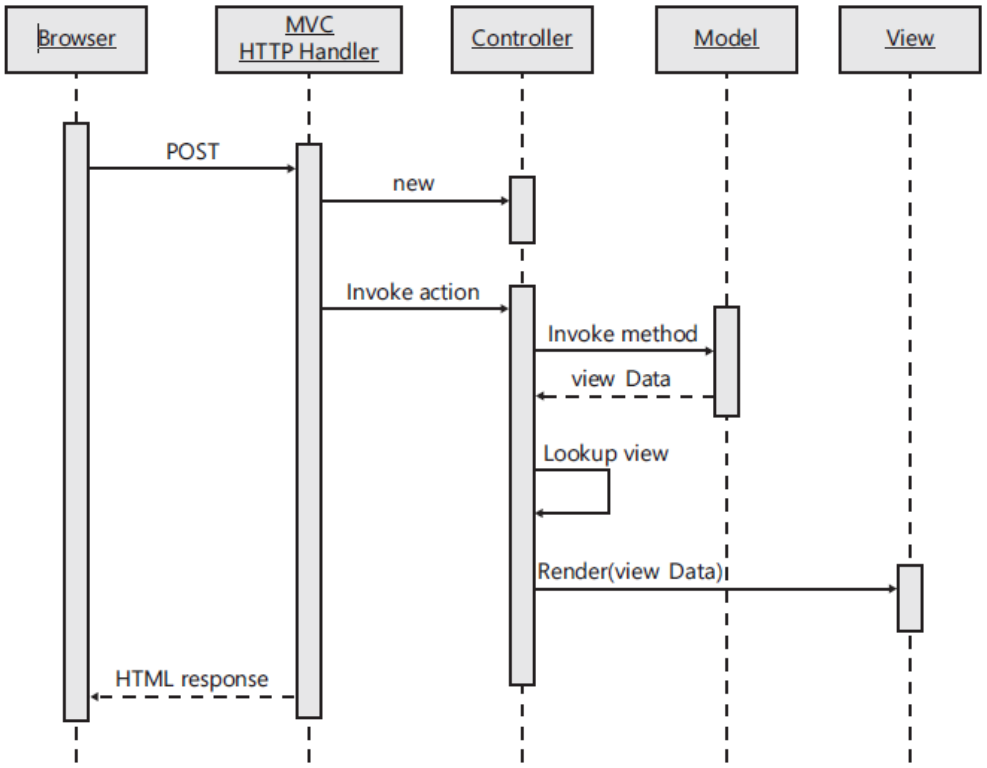

Figura 9 - ASP.NET MVC y el patrón Model2

La diferencia entre el patrón MVC y Model2 MVC. Como se puede ver en la figura, es que no hay contacto entre la vista y el modelo como en la formulación MVC original. La acción del usuario no es capturada y manejada por la vista. El controlador representa la vista y pasa explícitamente los datos de pantalla a la misma.

3.2.2 El Modelo

Los objetos del modelo son las partes de la aplicación que implementan la lógica de dominio de datos. Los objetos del modelo tienen la función de consultar, modificar y almacenar los manejados en la aplicación en una base de datos. Por ejemplo, un objeto al que llamaremos producto puede recuperar información de una base de datos, trabajar con él y hacer modificaciones y luego escribir la información actualizada en una tabla Productos en una base de datos SQL Server. (ASP.NET, 2012)

El modelo no es mas que la abstracción de modelos de datos (generalmente) bases de datos dentro del proyecto, es esto lo que se puede encontrar en la carpeta Model de un proyecto MVC. Esta conexión o relación se puede lograr gracias a una serie de elementos que provee ASP.NET y que estan disponibles tanto en modelo de desarrollo Web Forms, como en MVC, estos elementos son los controles de origen de datos, los que se pueden observar con mas detalle en las sección 3.1.2 de este trabajo, llamada Acceso a bases de datos. Aunque existen variedad de controles de origen de datos los mas utilizados son el LinqDataSource y el EntityDataSource, ya que estos dos proveen funcionalidades que permiten fácilmente realizar consultas y hacer operaciones de creación, actualización y eliminación de registros, además de proveer herramientas para realizar ordenamiento y filtrado de datos.

A modelo de datos de una aplicación MVC, lo llamamos dominio de la capa de negocio. Utilizando los controles de origen de datos, se crean una serie de clases dentro del proyecto que nos representan dicha capa y con la que podemos interactuar dentro de todo el proyecto, es por esto que dentro de ASP.NET MVC el termino modelo puede tener tres significados diferentes:

- **La representación de las entidades de dominio específico que operan en la capa de negocio**

En una aplicación MVC se debe tener una representación coherente de los datos de los procesos de aplicación. Estos datos se deben describir las entidades que hacen parte el dominio de la aplicación. Por ejemplo, una aplicación que trata de una empresa comercial probablemente tendrá entidades como cliente, pedido y la factura.

En una aplicación Web, los tipos de datos definidos en el nivel de negocio de la aplicación se elevan hasta la capa de presentación, en el que puede ser consumido por los controladores. Así, por ejemplo, si la entidad del modelo de datos se basa en un objeto de cliente, el objeto cliente mismo podría llegar a ser visible para el controlador y, desde allí, se puede pasar a la vista para concertar una página Web. En el camino de vuelta, el contenido de la forma puede ser dirigido de nuevo a un objeto de cliente en el contexto de

un método de acción del controlador y, desde allí, hasta la capa de negocio para cerrar el círculo. (Esposito, Programming Microsoft ASP.NET MVC, 2010)

- **La representación de los datos a ser enviados al controlador**

ASP.NET MVC funciona sobre el mismo entorno de ejecución clásico ASP.NET. Esto significa que todas las solicitudes ASP.NET MVC se manejan igual que en modelo clásico es decir, mediante un POST. Debido a esto el método de un controlador fácilmente puede agarrar las colecciones de datos enviados en los formularios a través del objeto Request, estos vienen sin una asociación u orden específico, se podría decir que vienen crudos. Por lo que ASP.NET MVC intenta modelar los datos entrantes en unas variables ya definidas en el dominio de la capa de negocio.

Por ejemplo en vez de hacer un request campo a campo en el formulario donde se están llenando los datos del cliente (customer), en MVC en un solo request se obtienen los datos del cliente, teniendo ya definida una variable cliente y colocándola como parámetro a recibir dentro de la acción como se muestra en la siguiente figura. (Esposito, Programming Microsoft ASP.NET MVC, 2010)

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Update(Customer customer)
{
    :
    :
}
```

.

Figura 10 - Representación de datos a ser enviados al controlador

En este caso la información enviada en el POST, automáticamente será convertida al tipo de dato Customer

- **La representación de los datos con los que trabaja la vista**

Después de que el controlador ha hecho su trabajo, es probable que se hayan producido, algunos datos para mostrar en la vista. Para mantener una separación limpia de las responsabilidades de cado uno, el controlador envía los datos a la vista. En otras palabras, la vista se espera que sea tan pasivo como sea posible y sólo se mostrará lo que recibe. El controlador le envía entonces a la vista una variable con un tipo específico de dato definido dentro del contexto del proyecto y la vista a su vez solo se encarga de presentar la información que le es enviada. (Esposito, Programming Microsoft ASP.NET MVC, 2010)

3.2.3 El Controlador

En los marcos de trabajo MVC, como Zend Framework, Ruby on Rails o ASP.NET MVC se suelen asignar direcciones URL en el código de servidor de una manera diferente. En lugar de asignar las direcciones URL de archivos físicos en el disco, se asignan direcciones URL directamente una serie de clases. Estas clases se denominan controladores y se encargan básicamente del manejo de las peticiones entrantes, del manejo de los datos entrados por el usuario, y de la ejecución apropiada de la aplicación.

En ASP.NET MVC los controladores se encargan de procesar las solicitudes de entrada del usuario y de ejecutar lógica de la aplicación correspondiente. Una clase de controlador posee una serie de acciones que se encargan de realizar una función determinada y de llamar a un componente de vista independiente para generar el código HTML que de respuesta a la solicitud.

La clase base para todos los controladores es la clase ControllerBase, que proporciona el manejo general de MVC. Cada clase Controller hereda las características de ControllerBase y es la implementación por defecto de un controlador. La clase de controlador es responsable de las siguientes etapas de procesamiento siguientes:

- Se encarga de localizar la acción o método adecuado para una solicitud. Se encarga de llamarla y de validar que si se le puede llamar.
- Obtener los valores que se utilizan como argumentos en la acción.
- Manejar todos los errores que pudieran ocurrir durante la ejecución de la acción
- Proporcionar la clase predeterminada WebFormViewEngine para representación de los tipos de de páginas ASP.NET (vistas)

- **Acciones**

En las aplicaciones ASP.NET que no usan el framework MVC, la interacción del usuario se organiza alrededor de las páginas, y en torno al manejo de eventos de la página y de los controles de la página. Por el contrario, la interacción del usuario con las aplicaciones ASP.NET MVC se organiza en torno a los controladores y a las acciones. El controlador define las acciones. Los controladores pueden tantas acciones como sean necesarias.

Las acciones suelen tener una correspondencia uno-a-uno con las interacciones del usuario. Algunos ejemplos de las interacciones del usuario incluyen: la introducción de una dirección URL en el navegador, el hacer clic en un vínculo, y el envío de un formulario. Cada una de estas interacciones del usuario provoca una petición que se enviará al servidor. En cada caso, la dirección URL de la solicitud incluye información que MVC utiliza para invocar un método de acción, la que también puede llevar uno o varios parámetros según se defina en el archivo global.asax y que ayudan a la aplicación a realizar acciones mas especificas.

La mayoría de las acciones devuelven una instancia de una clase que deriva de ActionResult. La clase ActionResult es la base de todos los resultados de la acción. Sin embargo, existen diferentes tipos de resultados de acción, dependiendo de la tarea que el método de acción se está realizando. Por ejemplo, la acción más común es llamar al método View. El método de vista devuelve una instancia de la clase ViewResult, que se deriva de ActionResult. (MSDN Library, 2012)

El siguiente cuadro presenta los diferentes tipos de resultados que en MVC puede dar una acción:

Tabla 2 - Tipos de resultados de acciones MVC

Resultado de la acción	**Método utilizado**	**Descripción**
ViewResult	View	Presenta una vista como una página web.
PartialViewResult	PartialView	Presenta una vista parcial, que se define como una sección o parte de otra vista.
RedirectResult	Redirect	Redirecciona a otra acción utilizando su dirección URL.
RedirectToRouteResult	RedirectToAction RedirectToRoute	Redirecciona a otra acción.
ContentResult	Content	Devuelve un tipo de contenido definido por el usuario.
JsonResult	Json	Devuelve un objeto JSON
JavaScriptResult	JavaScript	Devuelve un script (secuencia de comandos) que se puede ejecutar en el cliente
FileResult	File	Devuelve una colección de datos binarios con los que se puede generar un archivo.
EmptyResult		Representa un valor de retorno en caso de que acción deba devolver un resultado nulo.

3.2.4 Las Vistas

Después de que la acción del controlador ha finalizado su trabajo, puede haber algunos datos para mostrar al usuario, estos datos son enviados por la acción del controlador a una vista. La mayoría de las veces, estos datos se fusiona a una plantilla HTML y se escriben en el flujo de salida. Las vistas se definen entonces como los componentes que muestran la interfaz de la aplicación de usuario (UI). Típicamente, esta interfaz de usuario se crea a partir de los datos del modelo. Un ejemplo sería una vista de edición de una tabla Productos que muestra cuadros de texto, listas desplegables y casillas de verificación basadas en el estado actual de un objeto Producto. (MSDN Library, 2012)

Una vista por lo general es un archivo de extensión cshtml para el caso de ASP.NET MVC en su version 3. Dicho archivo se se encuentra en una carpeta con el mismo nombre de un controlador y lleva por nombre la acción que lo ejecuta. Por ejemplo la acción Index del controlador Home esta asociada con un archivo Index.cshtml que se encuentra en la capeta Views, subcarpeta Home. La vista no es mas que codigo HTML mezclado con código de un motor de vista, para el caso de este trabajo vamos a hablar del mas popular de ellos: razor. En ASP.NET MVC las vistas estan apoyadas en un Layout que cumple la misma función que la Master Page en Web Forms, para ver su definición favor ver la sección 3.1.3 Presentación de datos en el subtitulo Master Pages.

- **Razor**

ASP.NET MVC siempre se ha apoyado el concepto de "motores de vista". Los motores de vista son los módulos conectables que implementan diferentes opciones de sintaxis en una plantilla HTML y que permiten realizar diferentes funciones de programación que ayudan a generar contenido HTML dinámico. Existen varios tipos de motores de vista entre los que se incluyen el tradicional que utiliza Web Forms de archivos .aspx y .ascx, y además han surgido también otros muy populares en ASP.NET MVC como lo son Spark y NHaml, pero el que más impulso ha tomado es Razor.

Razor posee las siguientes características:

- Compacto, expresivo y fluido: Razor minimiza significativamente el número de caracteres de un archivo, en comparación con sus antecesores y permite un flujo de trabajo rápido y fluido de codificación. A diferencia de la mayoría de sintaxis de plantilla, no es necesario interrumpir su codificación para indicar explícitamente los bloques de código de servidor dentro del código HTML. El analizador es lo suficientemente inteligente para deducir esto de su código. Esto permite una sintaxis mucho más compacta y expresiva que hace de razor un lenguaje limpio, rápido y divertido de escribir.

- Fácil de aprender: Razor es fácil de aprender y permite ser productivo rápidamente con un mínimo de conceptos.

- No es un nuevo lenguaje: Cuando se creo Razor se decidió no crear un nuevo lenguaje. En su lugar, quería que los desarrolladores puedan utilizar sus conocimientos actuales en C # / VB (u otro) y aplicarlos en Razor.

- Posee Intellisense: Razor ha sido diseñado para no requerir una herramienta específica o un editor de código, pero si se trabaja con Visual Studio posee el apoyo de intellisense. (Guthrie, 2010)

- **Vistas Parciales**

Cuando se desarrolla en el marco de desarrollo ASP.NET MVC, naturalmente, se necesita crear componentes reutilizables de código para las vistas. En el modelo Web Forms, se podía hacer esto creando un control de usuario web o un control de servidor web, pero en MVC estos conceptos no existen y se deben utilizar las vistas parciales. (Mikesdotnetting, 2009)

Vistas parciales en ASP.NET MVC 3 le permiten al desarrollador crear contenido reutilizable con la sintaxis Razor de una manera sencilla. La sintaxis que representa la vista parcial se implementa como un archivo auxiliar HTML. Mediante la instrucción Html.Partial dentro de una vista cualquiera, se embebe el contenido de la vista parcial dentro de la vista. Esto es muy útil para adicionar menús, controles de sesión, pies de página o simplemente para posicionar información que se repite continuamente en varias vistas evitando así la duplicación del código.

3.2.5 Seguridad: Autenticación y autorización

ASP.NET MVC soporta todas las características de seguridad que posee el marco de desarrollo Web Forms, como lo son el Membership Provider, la autenticación por formularios, la autenticación Windows, entre otras. Estas funcionalidades se pueden observar en la sección 3.1.4 Seguridad. Pero además de esas características el modelo MVC provee nuevas funcionalidades como lo son:

- Autorización: permite tener un mayor control en la autorización de acceso determinados sitios de la aplicación. En la definición de cualquier acción de un controlador se tiene la opción de forzar la autenticación mediante el uso de la opción “Authorize”, en la que se puede indicar que usuarios o que roles de la aplicación pueden tener acceso a dicha acción, de no tener permiso se le deniega automáticamente el acceso.

- Prevención de ataques XSS: ASP.NET MVC previene automáticamente que los usuarios ingresen códigos maliciosos, generalmente código JavaScript o código HTML en los campos de entrada de los formularios. Cuando se trata de ingresar este tipo de código se deniega el acceso. ASP.NET también provee una biblioteca de AntiXSS que es de fundamental ayuda para prevenir este tipo de ataques. XSS (Cross-site scripting) es una técnica común de ataque a aplicaciones web con la que se quiere hacer daño a la aplicación, haciendo que de alguna manera falle, accediendo a sitios no autorizados, robando información confidencial, suplantando usuarios, etc.

- Prevención de ataques CSRF: Los ataques CSRF (Cross-site request forgery - falsificación de petición en sitios cruzados) Un ataque CSRF fuerza al navegador

web validado de una víctima a enviar una petición a una aplicación web vulnerable, la que entonces realiza la acción elegida a través de la víctima. Al contrario que en los ataques XSS, los que explotan la confianza que un usuario tiene en un sitio en particular, el *cross site request forgery* explota la confianza que un sitio tiene en un usuario en particular. (Wikipedia, 2012) MVC logra prevenir este tipo de ataques utilizando en las acciones POST una opción llamada AntiForgeryToken

3.3 ANÁLISIS DE SIMILITUDES Y DIFERENCIAS

3.3.1 Aspectos comunes de ambos modelos

Según lo investigado en los numerales anteriores de este trabajo donde se trataron las principales características de cada uno de estos modelos de desarrollo se puede concluir que estos tienen similitudes en los siguientes aspectos:

1. Entorno de tiempo de ejecución: ambos soportan su ejecución y compilación en .NET Framework, entonces por ende se apoyan en el CLR y utilizan las bibliotecas de clases de .NET para realizar las principales funciones, como lo son el acceso a bases de datos, acceso a archivos, envió de correos electrónicos, manejo de colecciones de datos, entre otras muchas poderosas herramientas que tiene la biblioteca de clases.

2. La ejecución en el IIS: Los dos modelos de desarrollo necesitan del IIS (Internet Information Services) para poder ejecutarse, es el IIS el que recibe las peticiones del usuario y las direcciona a las aplicaciones.

3. Seguridad: En los dos marcos de trabajo se puede utilizar Membership Provider, Forms Authentication, Windows Authentication y Passport Authentication.

4. Los controles de origen de datos: Los que se utilizan en Web Forms, que son los de ASP.NET en general, se pueden utilizar también en MVC para generar el modelo de datos.

5. El uso de una página maestra (Master Page/ Layout): El funcionamiento de esta característica es similar en los modelos de programación y su función es la misma proporcionar un esquema único para todas las interfaces de usuario de la aplicación web y evitar así la duplicación innecesaria de código.

3.3.2 Diferencias entre los modelos de desarrollo

Al ser ASP.NET MVC la respuesta de Microsoft a las falencias que presenta Web Forms debido a que ya lleva aproximadamente 10 años como el modelo de programación en la web de .NET Framework y debido a que las tendencias del desarrollo web han cambiado

significativamente, las diferencias entre los dos modelos son muy grandes y las podemos resumir así:

1. El patrón de diseño que las soporta: Web Forms esta basado en el patrón Page Controller, mientras que MVC se basa en la combinación de los patrones Front Controller y Model2.

2. La lógica de desarrollo: Web Forms es un modelo orientado eventos, diferente a MVC donde la lógica esta inspirada en la separación de conceptos y responsabilidades entre el modelo, el controlador y la vista.

3. El ciclo de vida de la página: En MVC el ciclo de la página es mucho mas sencillo que en Web Forms. En el primero el ciclo consiste en una petición que va al MVCHandler y este a la acción correcta dentro de un controlador y la acción a su vez devuelve el contenido HTML al navegador. En el caso de Web Forms el ciclo de vida pasa por una serie de eventos descritos en mejor detalle en la sección 3.1.1 Funcionamiento de Web Forms, subtítulo ciclo de vida de la página.

4. En MVC no existe el archivo de *codebehind*, por lo que las vistas no tienen una lógica que este asociada directamente a ellas, claro que en cierto modo su función la remplaza el controlador.

5. No existe el *postback* en MVC

6. No existe el *view state* en MVC

7. El manejo de las URLs: En MVC las URLs son lógicas representan una acción del controlador, además de esto, las URLs son flexibles y personalizables. Mientras que en Web Forms una URL representa un archivo físico.

8. El uso de controles de servidor en las páginas no es compatible con MVC y por consiguiente pierde sentido

9. En MVC se controla más fácilmente el código HTML generado, aunque en Web Forms también se podría llegar a controlar pero esto depende más del diseño que de la tecnología en si misma.

   .

10. MVC permite el TDD (Test Driven Development) por ser un modelo de desarrollo orientado a pruebas

11. MVC requiere un nivel de conocimiento más alto tanto del modelo de programación y de los patrones de diseño, como de lenguajes como HTML y JavaScript. Su curva de aprendizaje es mucho más difícil que la de su antecesor.

3.3.3 Paralelo

La siguiente tabla se resume en las principales propiedades de ambos modelos de desarrollo.

Tabla 3 - Paralelo Web Forms vs MVC

Aspecto	**Web Forms**	**MVC**
Patrón de diseño	Page Controller	Front Controller + Model2
Ciclo de Vida de la página	Apoya su funcionamiento en el IIS Cuando la página es procesada pasa por una serie de eventos similares al proceso de carga de un formulario Windows (Ver ciclo de vida de la página Sección 3.1.1)	Apoya su funcionamiento en el IIS Utiliza un objeto MVCHandler que se encarga de con base a la URL ejecutar el controlador y la acción que se solicitaron.
Lógica de desarrollo	Orientada a eventos. Enfocada en RAD (Desarrollo rápido de aplicaciones)	Separación de responsabilidades entre el modelo, el controlador y la vista
URLs	Referencia a archivos físicos aspx	Direccionamiento lógico. Bajo el esquema controlador/acción/parámetro. Es personalizable.
Seguridad	Permite el uso de Membership Provider y de las autenticaciones por formulario, Windows y Passport.	Permite el uso de Membership Provider y de las autenticaciones por formulario, Windows y Passport. Ayuda a prevenir ataques XSS y CSRF
Conexión a bases de datos	Se utilizan los controles de origen de datos de ASP.NET	Se utilizan los controles de origen de datos de ASP.NET
Realización de pruebas	Permite la realización de pruebas, pero es	Es un modelo de programación que fue

	extremadamente difícil realizarlas en la interfaces de usuario.	diseñado para permitir TDD.
Interfaz de usuario	Esta compuesta por código HTML y controles de servidor	Compuesta por HTML y código de un motor de vistas, por ejemplo Razor.
Dificultad de adaptación al modelo de desarrollo	Es relativamente sencillo trata de emular el desarrollo de aplicaciones Windows.	Requiere un buen nivel de conocimiento del modelo de programación. Se necesita conocer lenguajes como HTML y JavaScript

3.3.4 Ventajas y Desventajas

ASP.NET MVC no es el sucesor de Web Forms. Es más bien una alternativa de alta calidad a Web Forms. Cada modelo tiene su propio conjunto de peculiaridades. En última instancia, es difícil, y también inútil, tratar de decidir cuál es objetivamente mejor. Elegir entre ASP.NET Web Forms y ASP.NET MVC es una cuestión de actitud personal, de habilidades, y por supuesto, los requisitos del cliente. Como arquitecto o desarrollador es necesario entender las diferencias estructurales entre los marcos para que usted pueda tomar una decisión reflexiva. (Esposito, Programming Microsoft ASP.NET MVC, 2010)

A continuación se enuncian las ventajas y desventajas de ambos modelos.

- **Web Forms**

ASP.NET Web Forms es una plataforma estable y madura respaldada por una serie de controles y herramientas desarrollados por terceros. El modelo Web Forms proporciona un entorno que imita de manera efectiva el desarrollo de aplicaciones de escritorio que obtuvo tanto éxito en el pasado con Visual Basic y las herramientas RAD. Como resultado, no se tiene que ser un experto en web con una gran cantidad de conocimientos de HTML y JavaScript para crear aplicaciones Web eficaces. La productividad y el desarrollo rápido de aplicaciones y prototipos han sido el sello que ha caracterizado a ASP.NET Web Forms.

ASP.NET Web Forms era perfecto para la época en que fue creado, pero años después, nos encontramos ante una serie de desafíos diferentes, y algunas de las características que originalmente eran fortalezas claras de ASP.NET ahora resultan ser puntos débiles. Por ejemplo, para las páginas web modernas, el hecho de que no se tenga un control muy estricto sobre el código HTML generado es un problema grave, ya que si este no es correcto dificulta la accesibilidad, la compatibilidad del navegador, y la integración con marcos de JavaScript populares como jQuery, Dojo, y PrototypeJS. Por otro lado el manejo erróneo que en ocasiones se hace del *postback* y el *view state* hacen que el volumen de información que fluye desde y hacia el servidor sea excesivamente grande y

castiga el rendimiento de la página. (Esposito, Programming Microsoft ASP.NET MVC, 2010)

- **MVC**

El modelo de desarrollo ASP.NET MVC presenta las siguientes ventajas:

- Clara separación de responsabilidades entre interfaz, lógica de negocio y de control, que además provoca parte de las ventajas siguientes.
- Generación de un código mas limpio y estructurado, totalmente independiente de la lógica de negocio. Esto facilita el trabajo en equipo, ya que mientras unas personas trabajan en las interfaces, otras se dedican de lleno a la lógica de la aplicación en los controladores.
- Facilidad para la realización de pruebas unitarias de los componentes, así como de aplicar desarrollo guiado por pruebas (TDD).
- URLs más amigables y entendibles.
- Simplicidad en el desarrollo y mantenimiento de los sistemas. El hecho de que los componentes se encuentren separados y bien estructurados simplificará las tareas de mantenimiento.
- Reutilización de los componentes.
- Facilidad para desarrollar prototipos rápidos.
- Sencillez para crear distintas representaciones de los mismos datos.
- Los sistemas son muy eficientes, y a la postre más escalables.

Y a su vez presenta los siguientes inconvenientes:

- El tener que acoplarse a una estructura predefinida, hace que a veces pueda incrementar la complejidad del proyecto. De hecho, hay problemas que son más difíciles de resolver, o al menos cuestan algo más de trabajo, respetando el patrón MVC.
- Al principio puede cierto esfuerzo adaptarse a esta filosofía, sobre todo a desarrolladores acostumbrados a otros modelos más cercanos al escritorio, como Web Forms.
- La distribución de componentes obliga a crear y mantener un mayor número de archivos. (Aguilar, 2010)

3.4 PROCEDIMIENTO DE MIGRACIÓN

3.4.1 Aspectos por tener en cuenta para pasar de Web Forms a MVC

- Conocimientos y experiencia del equipo de desarrollo: Web Forms permite el desarrollo rápido de aplicaciones (RAD). A través de diseñadores visuales permite crear una página compleja con una relativa facilidad, ya que Web Forms se encarga del trabajo duro, como el mantenimiento del estado de las peticiones y el convertir las propiedades de los controles de servidor en código HTML, CSS y JavaScript. No se puede olvidar que para un determinado tipo de aplicaciones Web Forms representa una buena opción, por lo tanto si el equipo de desarrollo tiene una gran experiencia trabajando con este modelo y no tiene unos buenos conocimientos de la programación a bajo nivel, ni experiencia previa trabajando con el patrón MVC se debería tener prudencia antes de dar el salto a ASP.NET MVC, puesto que la productividad va a caer, al menos inicialmente.

- Necesidad de usar controles o sistemas prexistentes: Otro aspecto que se debe tener en cuenta antes de dar el salto de Web Forms a MVC, es que existe una probabilidad muy grande de no poder usar porciones de código o componentes visuales desarrollados previamente. Allí no funcionarán los controles de servidor, ni las plantillas de proyecto, ni los generadores de código, sin embargo cada día existen más componentes para ASP.NET MVC de compañías dedicadas a la creación de herramientas de programación y otros generados desde comunidades de desarrolladores que ayudaran a hacer mucho mas productivo el desarrollo. También es de anotar que ASP.NET MVC fue hecho para ser compatible con las librerías JQuery y su interminable cantidad de complementos donde se puede encontrar solución a prácticamente cualquier necesidad que se tenga para la interfaz del usuario.

- El uso del postback y el viewstate: en estos dos conceptos se basa el funcionamiento de ASP.NET Web Forms y en el caso del view state, el mecanismo que permite mantener persistente el estado de los controles, puede ser difícil migrarlo a MVC

- La importancia de las ventajas propias de MVC: Las ventajas del modelo MVC descritas anteriormente en este trabajo son un buen aliciente para trabajar con ASP.NET MVC, de hecho se debería tener en cuenta que aspectos del desarrollo se van a mejorar con la adopción de esta tecnología y si esas ventajas compensan los inconvenientes que puede generar el cambio.

- El tipo de sistema que se piensa construir: A la hora de plantearse el cambio de Web Forms a MVC es importante pensar en que tipo de proyecto se esta trabajando. No es lo mismo trabajar en un sitio web colaborativo destinado a un gran número de usuarios, donde el control de entradas y salidas es crucial para asegurar aspectos como la escalabilidad, el cumplimiento de estándares o la accesibilidad, que trabajar en un aplicativo de gestión que utilizará un grupo relativamente reducido de usuarios en una intranet corporativa. En ambos casos se trata de sistemas web, pero los objetivos, requisitos y restricciones a considerar son bien diferentes. Para el primer caso ASP.NET MVC es la mejor opción, la

simplicidad de MVC hace que el ciclo de vida de las páginas sea mucho mas sencillo que el de Web Forms y la ausencia de automatismos y la persistencia del estado aligera en gran medida el peso y complejidad de las páginas, lo que redundará muy positivamente en el rendimiento de la página. En cambio, el segundo caso, cuando se trata de crear pesadas aplicaciones de gestión con interfaces de usuario complejas y en las que no es especialmente relevante la calidad del código HTML enviado al cliente, ni el rendimiento. (Aguilar, 2010)

3.4.2 Migración de Aplicaciones

En informática cuando nos referimos a procedimientos de migración, generalmente se crea la errónea idea de que se trata solo de bases de datos, pero una migración también puede ser de aplicaciones de software cuando estas se vuelven obsoletas y necesitan ser actualizadas a una nueva tecnología.

Una migración es un procedimiento que se realiza para mover o trasladar los datos almacenados en un origen de datos a otro, de un sistema a otro, de un sistema operativo a otro, etc. Pero para el caso de la migración de aplicaciones consiste en mover un conjunto de instrucciones o programas de una plataforma a otra, reduciendo al mínimo la reingeniería, encontrando equivalentes paralelos entre las plataformas y convirtiendo las entradas, salidas y archivos a los formatos deseados. La migración puede ser simplificada con ayuda de herramientas que permiten convertir el código de una plataforma a otra, bien sea código compilado o interpretado (Menendez Soto & Robledo Fong, 2002). Pero en muchas ocasiones es necesario hacer una reescritura manual de todo el código de la aplicación, casi siempre se da porque se trata de aplicaciones o lenguajes legados y la migración requiere un lenguaje de programación muy diferente al original.

La migración de aplicaciones es una de las tareas mas difíciles que se llevan a cabo en el campo de la computación, es por esto que es indispensable que antes de empezar cualquier proceso de esta naturaleza, se tenga clara la razón por la cual se está migrando, además de elaborarse la planeación detallada, ya que de lo contrario se estarían corriendo unos riesgos muy altos de fracasar.

Para el desarrollo de este trabajo se estuvieron estudiando varias metodologías de migración, la mayoría de ellas provienen de empresas de consultoría cuyo foco de negocio esta en este tipo de migración. Todas ellas coinciden en algo: se debe tener muy organizado y planeado todo el proceso, analizando el alcance y proponiendo metas claras y realizables. A continuación se procederá a exponer tres de estas metodologías que ayudaron a plantear la metodología de migración para este trabajo, claro esta que estas metodologías están pensadas en la migración de software y datos en general:

1. **Metodología de IT Consultancy Services:**

   La metodología de IT Consultancy Services se basa en 5 pasos con sus respectivas actividades. (Para un mejor entendimiento ver figura 11).

- Inventariado: Revisa e identifica las clases, formularios y módulos del sistema a migrar.

- Análisis: Se analiza la estrategia de migración, analiza el sistema y su comportamiento con la nueva tecnología que va a usar, identifica las limitaciones de la nueva plataforma respecto a la anterior y sugiere estrategias para evitarlas

- Diseño y prototipo: Se realiza un prototipo para un mejor entendimiento e identificación de los temas de la migración. Se diseña un proceso que permita tener claridad en el manejo que se le da al problema. Se preparan listas de chequeo.

- Codificación y pruebas unitarias: En este punto se realiza la migración utilizando bien sea herramientas automáticas o realizando la migración de una forma manual. Y a la vez se hace una revisión para asegurar la finalización y el éxito del procedimiento

- Integración y pruebas: Paso final en el que se pone a prueba el comportamiento de la aplicación en un ambiente real. (IT Consultancy Services, 2012)

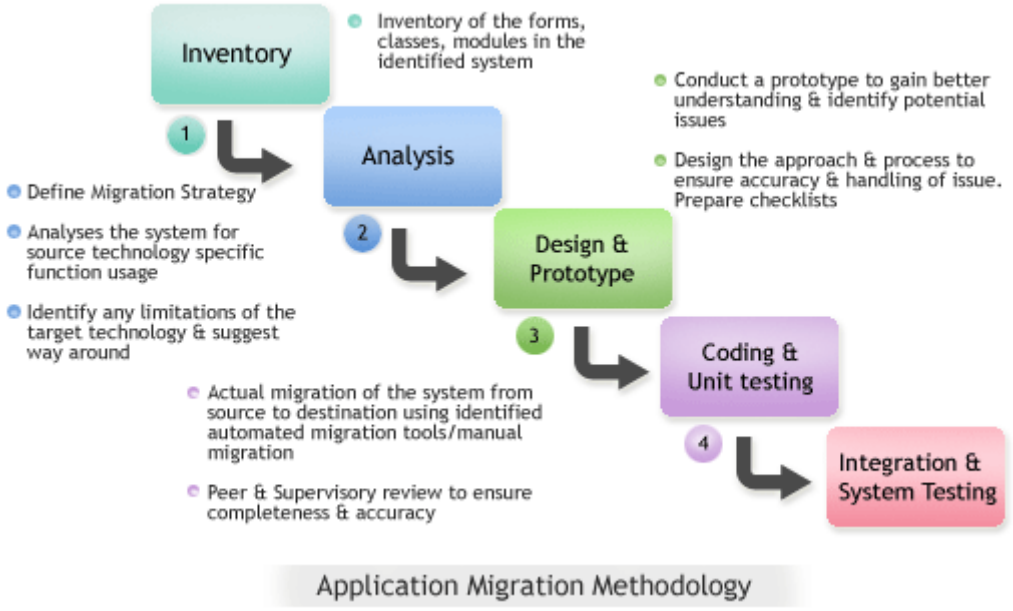

Figura 11 - Metodología IT Consultancy Services

2. Metodología Rapidsoft Systems

Rapidsoft Systems ha establecido un programa de migración que ofrece un proceso rápido y seguro de transferir aplicaciones estratégicas para a un nuevo entorno, este proceso esta divido en los siguientes pasos:

- Análisis: Análisis y estudio de viabilidad inicial de la arquitectura de la aplicación, las dependencias de código fuente y creación del ambiente de desarrollo; estimación en tiempo y recursos de la migración a la plataforma moderna

- Valoración: Examinación del código de la aplicación a migrar basada en sistemas operativos, el entorno de ejecución, y la interacción de la aplicación con otro software, el nivel de esfuerzo estimado y plan de migración inicial.

- Implementación: En este paso se lleva a cabo la transformación del código, la construcción del ambiente de desarrollo y la integración con otras aplicaciones, el diseño de base de datos y la conversión de los mismos para que sean compatibles y la evaluación comparativa de rendimiento.

- Migración: En esta etapa la migración del sistema se culmina, los sistemas se pueden ejecutar en paralelo, los usuarios deben ser informados y entrenados en los cambios que tuvieron lugar, preferiblemente por medio de documentación de usuario. (Rapidsoft Systems, 2012)

3. Metodología de Inteq

La empresa Inteq presenta la siguiente metodología. (Ver figura 12 – Metodología Iteq)

- Análisis: Valoración y planeación.

- Diseño: Ingeniera inversa y diseño detallado.

- Migración y pruebas.

- Aprobación: Validación e implantación. (Inteq, 2012)

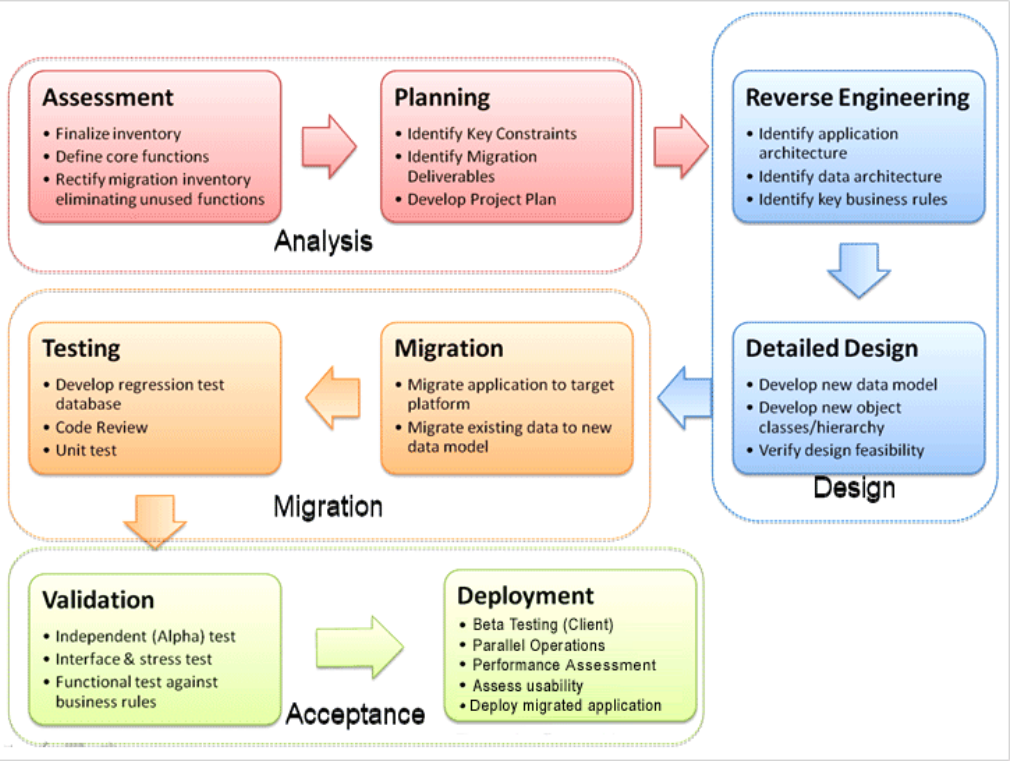

Figura 12 - Metodología de migración Inteq

3.4.3 Ejemplo de Migración

En este trabajo, por medio de un ejemplo documentado, se va a exponer un procedimiento para la migración de aplicaciones desde el modelo de desarrollo ASP.NET Web Forms a ASP.NET MVC. Este procedimiento se hace con base en la información recopilada en los anteriores capítulos de este proyecto, sobre ambos modelos de desarrollo y sobre la migración de aplicaciones.

El ejemplo a documentar es una aplicación web llamada Tailspin Spyworks, dicha aplicación es un tutorial provisto por la página oficial de documentación sobre ASP.NET de Microsoft sobre ASP.NET Web Forms. Dicho tutorial se puede encontrar en la siguiente dirección: http://www.asp.net/web-forms/tutorials/tailspin-spyworks. Tailspin Spyworks es una aplicación básica de Web Forms que emula el comportamiento de una página de compras en línea, donde se pueden consultar productos, realizar compras (a través de la figura del carrito de compras), consultar compras anteriores, etc. En el desarrollo de esta sección se entrará a analizar más a fondo las funcionalidades de este aplicativo. Es de anotar que tanto la aplicación desarrollada en Web Forms como su migración a MVC que son proyectos de Visual Studio 2010, y la base de datos utilizada se podrán encontrar en un CD adjunto a este trabajo

Un procedimiento como este debe tener una metodología clara a seguir, que garantice el éxito y minimice los riesgos que se puedan tener. Es por esto que teniendo en cuenta las metodologías descritas anteriormente en la sección 3.4.2 y el contexto de la migración (es decir que se trata de una migración de modelo de desarrollo, no de base de datos, ni de plataforma ya que sigue siendo ASP.NET), para el ejemplo de migración de este trabajo se van a seguir estas fases: análisis, planeación, desarrollo e implementación. Cada fase tiene un objetivo concreto que esta descrito en la siguiente tabla:

Tabla 4 - Fases migración de Web Forms a MVC

Fase	Objetivo
1. Análisis	Definir el proyecto precisando las metas, alcance, restricciones y suposiciones.
2. Planeación	Definir de la especificación funcional y plan del proyecto.
3. Desarrollo	Realizar el desarrollo, prueba y creación del sistema.
4. Implementación	Liberar el nuevo sistema, evaluar el desempeño y corregir los problemas que se presenten.

A continuación se va a realizar una especificación funcional de la aplicación Tailspin Spyworks en Web Forms y después se describirá con detalle las actividades llevadas a cabo en cada fase para realizar el proceso de la migración.

- **Tailspin Spyworks**

  1. **Principales funcionalidades**

     Tailspin Spyworks es una aplicación sencilla de una tienda en línea. En esta tienda en línea basa su funcionamiento en los siguientes tres conceptos:

     - Producto: En la aplicación existen una serie de productos, cada producto pertenece a una categoría específica. Un producto tiene un nombre, una descripción, un precio y unos comentarios. El usuario puede ver el detalle de cada producto y a la vez puede ingresar comentarios acerca del mismo, siempre cuando se encuentre autenticado. También puede agregar productos al carrito de compras.

     - Carrito de compras: Un carrito de compras es un lugar temporal en donde se van guardando el listado de los productos que el usuario desea comprar con sus respectivas cantidades. El usuario puede consultar en el momento que desee su carrito de compras y actualizarlo, es decir modificar cantidades, eliminar productos o agregar nuevos. El usuario puede utilizar el carrito de compras sin estar autenticado pero al momento de realizar la compra debe autenticarse.

- Orden: La orden es el registro de una compra realizada por el usuario, es decir cuando el acepta comprar los productos de su carrito de compra, este se convierte en una orden. Esta no se puede modificar, solo es posible consultarlo.

En general la aplicación Tailspin Spyworks permite al usuario realizar las siguientes funcionalidades:

- Autenticarse
- Cerrar sesión
- Consultar listado completo de productos
- Consultar listado de productos por categoría
- Ver el detalle de un producto
- Realizar comentario o revisión de un producto
- Agregar producto a carrito de compras
- Ver carrito de compras
- Borrar un producto de carrito de compras
- Actualizar la cantidad de un producto en el carrito de compras
- Realizar compra
- Ver historial de ordenes (compras realizadas) y su detalle
- Ver los productos mas comprados

La siguiente figura ilustra la estructura de la interfaz de usuario de la página en la funcionalidad de consulta del carrito de compras

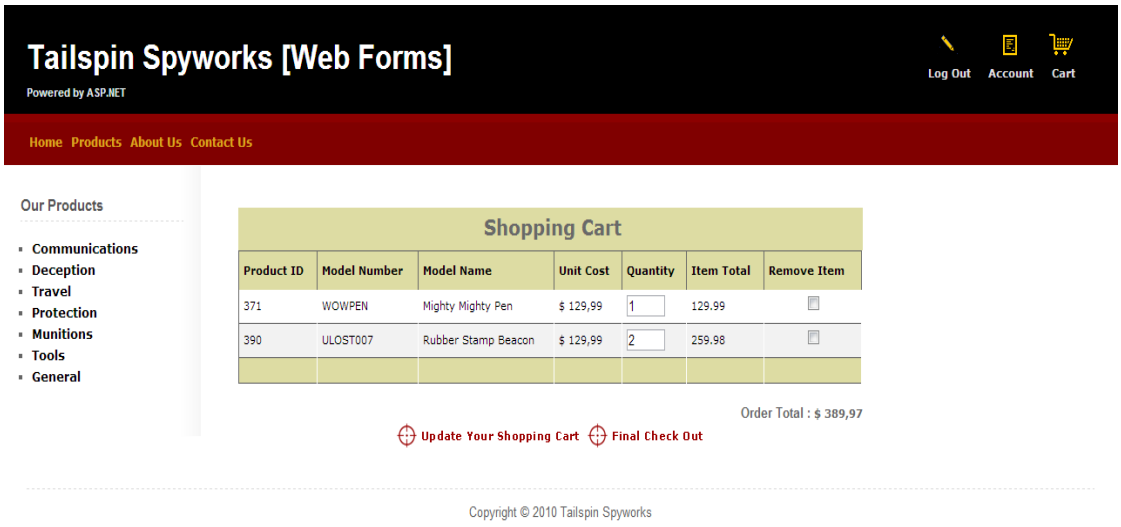

Figura 13 - Carrito de compras Tailspin Spyworks Web Forms

2. Mapa del sitio

El mapa de navegación de la página Tailspin Spyworks con todos los formularios web que se muestran al usuario se presenta la figura 14, se va a describir seguidamente los datos que muestran y las funcionalidades de cada uno de ellos. Es importante decir que cada uno de estos archivos tiene un archivo *codebehind* que regula su comportamiento ante determinados eventos.

- Default.aspx: es la página de inicio de la aplicación, en esta interfaz se listan todas las categorías y los productos que mas han sido comprados.

- AboutUs.aspx: esta página solo muestra contenido estático referente a la tienda en línea.

- ContactUs: presenta un formulario de contacto, el que envía la sugerencia hecha por un usuario al correo del administrador de la página

- Login.aspx: formulario para presentar las credenciales y autenticarse. En caso de estar autenticado se cierra la sesión.

- OrderList.aspx: muestra el listado todas las compras u ordenes realizadas por el usuario. Desde esta página se puede ir a ver el detalle de una compra. Para ver esta interfaz es estrictamente necesario estar autenticado.

- OrderDetails.aspx: muestra el contenido de una compra u orden particular.

- MyShoppingCart.aspx: desde esta vista se puede ver el detalle del carrito de compras. Aquí el usuario tiene la posibilidad de actualizar el carrito de compras ya sea eliminando productos y/o modificando las cantidades, también es desde este punto donde se finaliza la compra (checkout). Para finalizar la compra es necesario estar autenticado.

- Checkout.aspx: es una vista confirmatoria del contenido de la compra, si se acepta el carrito de compras se convertirá oficialmente en una compra y quedará en el historial. Solo usuarios autenticados acceden a este punto.

- ProductList.aspx: en caso de no tener parámetro, muestra todos los productos registrados en la base de datos y si lleva un parámetro (que es el identificador de una categoría) se visualizan los productos por categoría. Desde esta interfaz se puede ver el detalle de cada producto.

- ProductDetails.aspx: el usuario puede observar el detalle del producto: nombre, descripción, precio y observaciones. Se puede desde aquí agregar el producto al carrito o se puede crear una observación.

- ReviewAdd.aspx: desde esta vista el usuario tiene la posibilidad de agregar una observación.

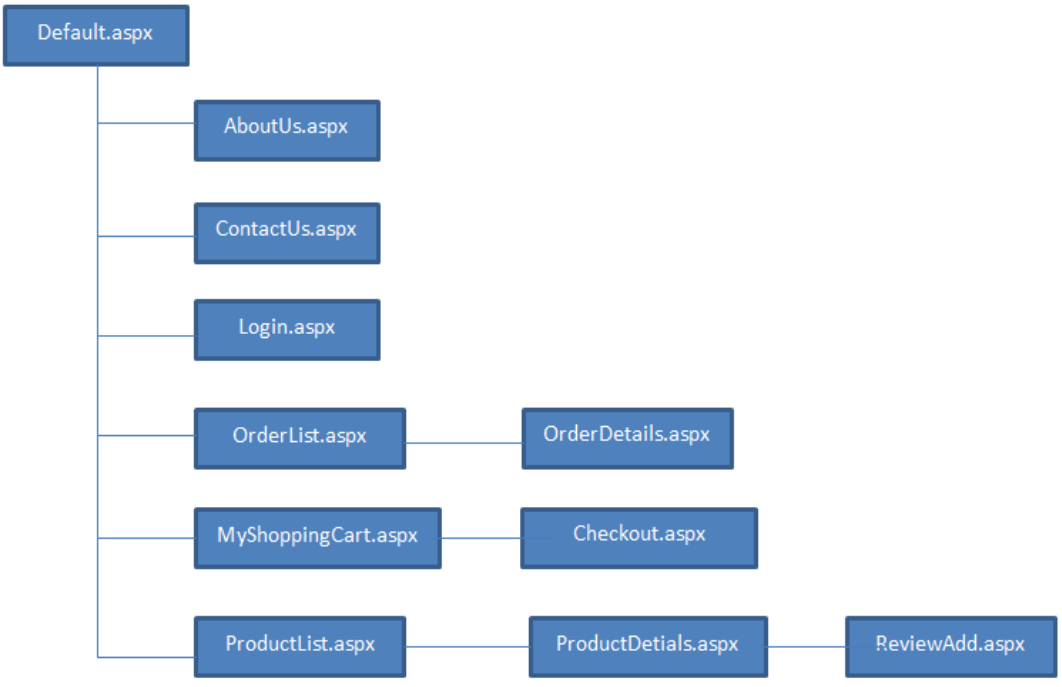

Figura 14 - Mapa de sitio Tailspin Spyworks

Además existe otro formulario que no se muestra al usuario que es AddToCart.aspx, este formulario solo tiene la labor del lado del servidor de

agregar un producto al carrito y direccionar la página al detalle del carrito de compras (MyShoppingCart.aspx).

3. Base de datos

Esta aplicación usa dos bases de datos para trabajar, la primera es la base de datos del Membership Provider que provee todos los recursos para la administración de los usuarios de la página. La segunda es la base de datos Commerce, que contiene todas las estructuras para guardar los registros transaccionales realizados sobre la aplicación. Se puede apreciar el diagrama en la siguiente figura.

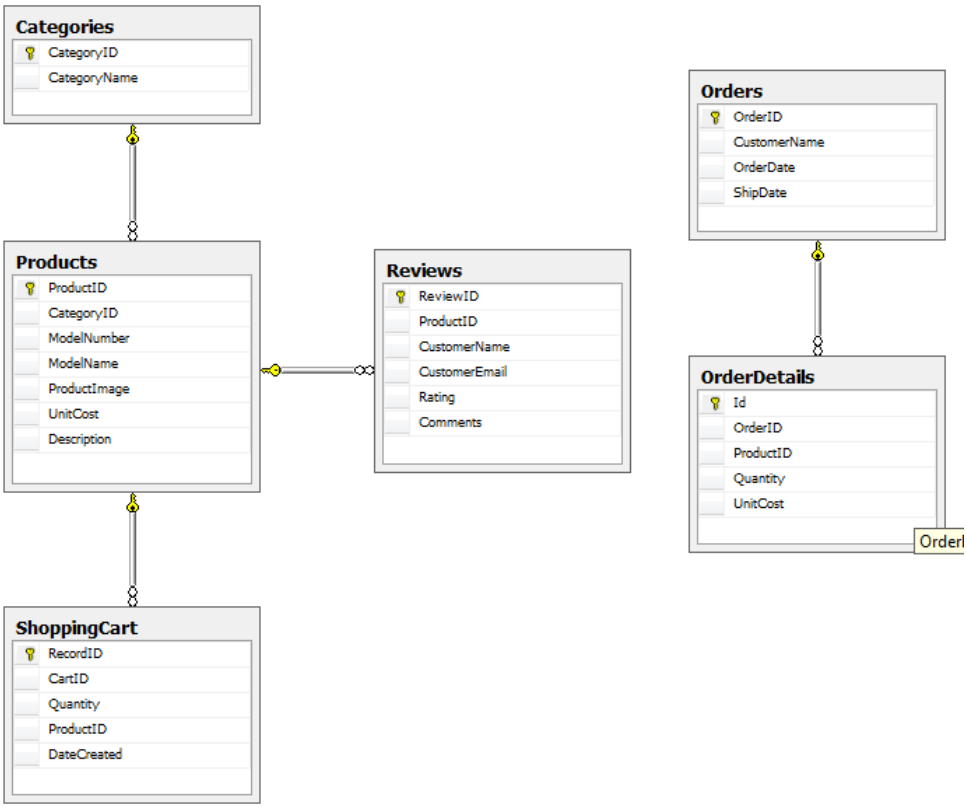

Figura 15 - Diagrama de base de datos Commerce de Tailspin Spyworks

La base de datos Commerce contiene las siguientes tablas:

- Products: Es la tabla donde se guarda la información de un producto. Un producto puede tener una o varias observaciones, así como también estar relacionado con varios registros de carrito de compras (ShoppingCarts)

- Categories: Almacena las categorías de los productos. Una categoría puede tener uno o varios productos.

- Reviews: Observaciones hechas a un producto
- ShoppingCart: Registro de carrito de compras.
- Orders: Guarda el registro de las compras. Una compra esta relacionada con uno o mas detalles de compra (OrderDetails)
- OrderDetails: Detalle de la compra.

○ Análisis

La migración desde el modelo de desarrollo ASP.NET Web Forms, hacia al modelo de ASP.NET MVC, no es algo sencillo, ni automatizado ya que estos difieren en su lógica de construcción en aspectos fundamentales, como ya se vio en la sección 3.3 de este trabajo. Pero también tienen cosas en común que son de gran ayuda para realizar la conversión, es que el solo hecho que modelos de desarrollo estén soportados sobre .NET Framework y que puedan ambos utilizar en lenguaje como C# o Visual Basic .NET para la lógica de negocio, representa un avance inmenso.

En esta fase de la migración se definió el contexto, las metas, el alcance y las limitaciones del proyecto. Para empezar ambos proyectos se ejecutan bajo la versión 4 de .NET Framework, se desarrollaron con el IDE Visual Studio 2010 y para el caso de proyecto MVC, este se trabajó sobre la versión de 3 de ASP.NET MVC, utilizando como motor de vistas a Razor. Se escogieron estas versiones y este motor de vistas ya que en momento de empezar este trabajo (segundo semestre de 2011) eran las más recientes.

La meta con esta migración es lograr pasar la aplicación, en este caso Tailspin Spyworks, completamente funcional al modelo ASP.NET MVC, sin que la navegación se vea afectada; el usuario final no debe notar a simple vista que se trata de otra aplicación diferente. Esto quiere decir que se deben reprogramar todas y cada de las funciones presentes para obtener el mismo resultado, sin realizar cambios en las bases de datos.

Es importante tener en cuenta las estructuras de cada uno de los proyectos ya que se debe adaptar el proyecto en Web Forms al modelo MVC, tener en cuenta que en MVC no existen los controles de servidor, ni el archivo *codebehind* de cada formulario. Se debe entonces para el caso de los controles de servidor observar el código HTML generado y emularlo y para el archivo *codebehind* acomodarlo en el controlador según las circunstancias.

Existen dos aspectos que no se van a migrar del proyecto Tailspin Spyworks el primero es el ASP.NET AJAX Control Toolkit que se encuentra en las formularios de contacto y de registro de comentarios, y que permite agregar fuente, tamaño de letra y otro tipo de ediciones al texto del comentario. Este complemento no es compatible con ASP.NET MVC y además no es de sustancial importancia para el funcionamiento del aplicativo. Tampoco se va a migrar la funcionalidad de envío de correo cuando se agrega una sugerencia en la página de “contáctenos” debido a que no se cuenta con un ambiente

con un servidor de correo disponible, lo que se hará es dejar todo configurado para que funcione pero no se podrán realizar pruebas.

- **Planeación**

En la migración de Web Forms a MVC hay que tener que el cambio de uno a otro modelo no es una simple conversión como podría ser pasar de una aplicación Visual Basic .NET a C#; se trata de un nuevo marco de trabajo que afecta sobre todo a la presentación y control de flujo del sistema. Si se tiene unas buenas clases de lógica de negocio, seguramente sean los únicos componentes que se puedan reutilizar de forma directa y sin grandes cambios. El resto, es decir, todo lo relativo a la interacción con el usuario y la lógica de control y navegación, hay que convertirlo de forma manual. (Aguilar, 2010)

Teniendo en cuenta lo anterior y los comparativos realizados en el desarrollo de este proyecto, para la migración de ASP.NET Web Forms a ASP.NET MVC se siguieron estos pasos:

1. Creación del proyecto MVC

2. Adición del modelo de negocio a la carpeta del modelo: este proceso se logra principalmente a través de los controles de origen de datos EntityDataSource o LinqDataSource. Se recomienda usar uno de estos controles ya que estos ayudan a automatizar las consultas, actualizaciones, inserciones y eliminaciones, además de que generan automáticamente las clases del modelo de negocio.

3. Relacionar base de datos del Membership Provider con el nuevo proyecto. (ASP.NET MVC provee un controlador por defecto llamado AccountController, este trae por defecto todos los procedimientos para trabajar con el Membership Provider)

4. Identificar controladores, acciones y vistas: De acuerdo al modelo de base de datos, al mapa de navegación del sitio y a las distintas funcionalidades presentes, visualizar cuales clases de la lógica de negocio son candidatos a controladores y cuales serian las acciones y vistas. Para con esta información realizar un diseño de la estructura del sitio en el modelo MVC.

5. Crear controladores, acciones y vistas de acuerdo con lo evaluado en el punto anterior.

6. Mover archivos estáticos, tales como imágenes o archivos CSS a la estructura de directorios correspondiente en el proyecto MVC.

7. Realizar la conversión de la Master Page de Web Forms al Layout de MVC. Teniendo en cuenta las diferencias de sintaxis que tienen y tratando de remplazar los controles de servidor utilizados allí con vistas parciales.

8. Desarrollar el proceso de conversión de cada formulario de Web Forms a la estructura Acción- Vista – Acción POST de MVC.

Generalmente un formulario de Web Forms esta compuesto por un archivo .aspx con toda la parte de la interfaz de usuario, incluyendo los controles de servidor y un archivo *codebehind*, para nuestro caso un archivo en C#. En el archivo *codebehind* se pueden encontrar varios métodos, nos interesa principalmente el método que se ejecuta cuando se carga la página (Page_Load) y el que se ejecuta cuando se envía algún formulario.

Se procede a hacer la conversión de la siguiente forma:

- Se crea la acción que va a remplazar el formulario, teniendo en cuenta que el procedimiento que se encontraba en el cargue del formulario y los parámetros, pasa toda esta parte al controlador. Si además dentro del archivo aspx existen controles de enlace de datos; la lógica que hace la consulta de estos datos pasa también a la acción del controlador, debido a que en el modelo MVC no se acceden a datos desde las vistas, cualquier dato que llega a la vista debe ser consultado por el controlador y este se lo transfiere a la vista.

- Luego se construye la vista a partir de la estructura del Layout y del contenido del archivo aspx, si este contiene controles de servidor se analiza el código HTML generado por estos y con base a este se remplazan los controles de servidor por código HTML dinámico, con la ayuda del motor de vistas Razor.

- En caso de que la vista tenga un formulario, se debe crear en el controlador la acción POST, el flujo de código que en Web Forms se encargaba evento de envío del formulario pasa a ser parte de esta acción.

9. Evaluar las funcionalidades de interacción con el usuario y de contenido dinámico de las diferentes interfaces de usuario, tales como ordenamiento, paginación, validación de datos, etc. Para luego determinar cuales se pueden convertir en funciones JavaScript, también valorar que librerías se podrían utilizar para lograr la misma funcionalidad; por ejemplo para las validaciones se podría pasar de los controles de validación de Web Forms a usar el complemento JQuery.validate.

10. Pruebas funcionales: Se prueba el correcto funcionamiento de la página, la navegación, las redirecciones, la presentación y envío de datos, etc.

o Desarrollo

En esta fase es donde se realiza la migración como tal del proyecto, con base en los pasos descritos en la fase de planeación. Se van entonces a enumerar a grandes rasgos las acciones que se realizaron para llevar a cabo la migración del proyecto Tailspin Spyworks de Web Forms. Para apreciar de mejor manera el procedimiento realizado, se puede consultar los archivos de los proyectos tanto el de Web Forms como el de MVC que son archivos digitales complementarios a este trabajo.

1. Creación del proyecto:

   Se creó un proyecto en ASP.NET MVC versión 3 con motor de vistas Razor en Visual Studio 2010.

2. Adición del modelo de negocio a la carpeta del modelo:

   En la carpeta Model del proyecto se creo un elemento de tipo LinqDataSource (LINQ to Sql Classes), también se podría hacer la conexión con otros controles de origen de datos como EntityDataSource. Se realiza luego la conexión a la base de datos Commerce en el Server Explorer (Administrador de conexiones de Visual Studio) y se arrastran todas las tablas al elemento LinqDataSource, generando así las clases del modelo de negocio que se utilizaron en el proyecto.

3. Relacionar base de datos del Membership Provider:

   El proyecto de Web Forms utiliza la base de datos ASPNETDB (base de datos que se encuentra en el servidor local sqlexpress CAMILO-PC\SQLEXPRESS) como fuente de información del Membership Provider. Para el proyecto de MVC lo que hacemos para que la página siga trabajando con los mismos usuarios es referenciar dicha base de datos en el archivo de configuración Web.config remplazando la siguiente instrucción:

   ```
   <add name="ApplicationServices" connectionString="data
   source=.\SQLEXPRESS;Integrated
   Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User
   Instance=true" providerName="System.Data.SqlClient" />
   ```

   Por esta:

   ```
   <add name="ApplicationServices" connectionString="Data Source=CAMILO-
   PC\SQLEXPRESS;Initial Catalog=ASPNETDB;User ID=tailspin;Password=see274"
   providerName="System.Data.SqlClient" />
   ```

   Respecto al funcionamiento del Membership Provider, ASP.NET MVC crea por defecto un controlador llamado AccountController que se encarga de la autenticación y la gestión de usuarios, por lo que el trabajo en esta migración solo representará la conversión de la interfaz de usuario, no tanto la lógica del controlador.

4. Identificar controladores, acciones y vistas:

   Con base en el mapa del sitio (figura 14), en el diagrama de base de datos de la aplicación (figura 15), en las diferentes funciones y en las interfaces de usuario. Se analizaron y dedujeron las figuras 16 y 17 (identificación de controladores, acciones y vistas) que representan, la conversión del modelo de Web Forms a la estructura MVC.

Estas figuras están separadas en tres partes: controladores, acciones y equivalencia con Web Forms. Cada controlador tiene sus acciones relacionadas, las acciones indicar varias cosas, si poseen una (V) quiere decir que representan una vista, si tienen el atributo (partial) representan una vista parcial y si después de su nombre se indica el nombre [POST] señala que es una acción POST de envío de datos de un formulario.

Surge entonces la pregunta de como se dedujeron estos diagramas, la respuesta se puede dar con un ejemplo: si se observa el formulario productDetails.aspx en Web Forms, se puede ver que presenta la información de un producto; por consiguiente en el modelo MVC representa una acción (llamada Details en la figura) que carga la información de un producto y la envía a una vista; hace parte entonces del controlador ProductsController. Luego observando la interfaz de usuario de esta página, vemos que aparte de la información del producto, se pueden ver las observaciones que se han hecho, por esta razón se hace necesario que la vista Products/Details se ayude de una vista parcial llamada _List (hace parte del controlador Reviews) que le presente el listado de observaciones de un producto. Además desde esta vista se puede agregar el producto al carrito de compras representado por la acción ShoppingCart/AddtoCart, la que es una acción POST. También desde esta vista se tiene la opción de crear una nueva observación, esta opción re direcciona a una acción que carga la interfaz para realizar esta actividad (Reviews/Add).

En las siguientes dos figuras se evidencia las decisiones tomadas para realizar la conversión de la página al modelo MVC.

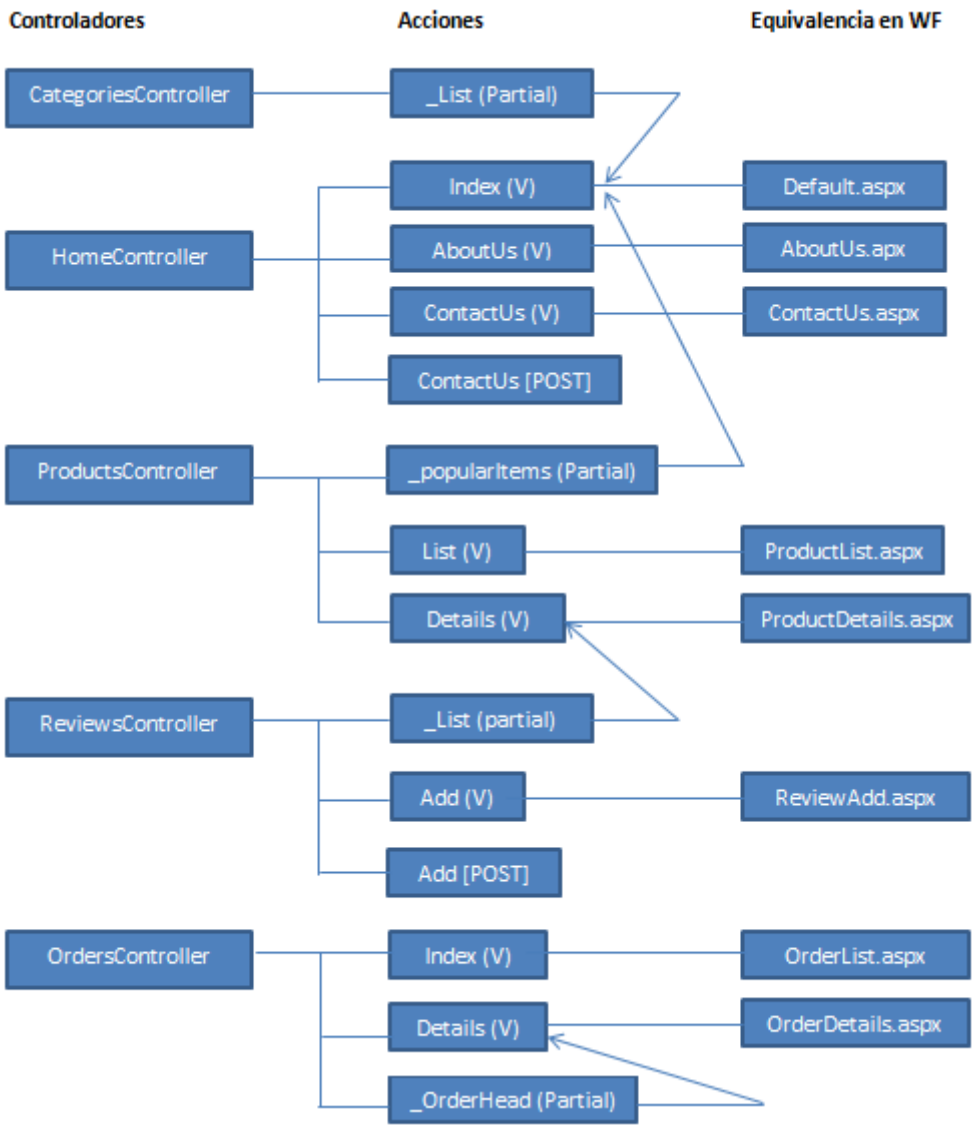

Figura 16 - Identificación de controladores, acciones y vistas (parte 1)

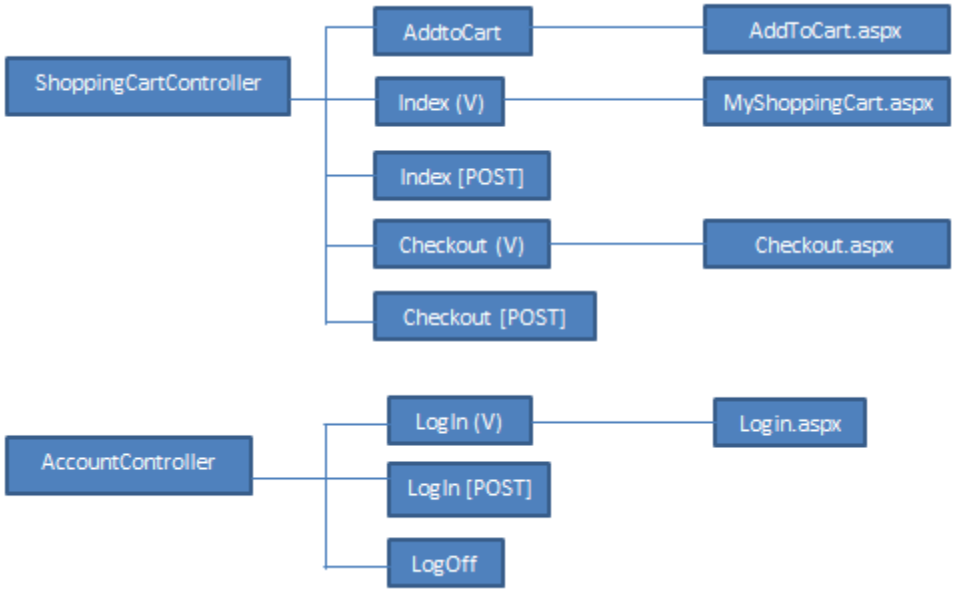

Figura 17 - Identificación de controladores, acciones y vistas (parte 2)

Es importante decir que por defecto un proyecto de ASP.NET MVC trae los controladores AccountController y HomeController, el primero encargado de la administración de usuarios; el segundo se ocupa de las acciones que no tienen un tipo de dato en la clase de negocio, como la página de inicio o la de contacto.

5. Creación de la estructura del proyecto:

   Se procede a crear la estructura definida en el paso anterior dentro del proyecto MVC, teniendo presente el tipo de acción a crear, los parámetros que requiere, los datos que recibe y que devuelve, los tipos de datos a manejar de cada vista y la seguridad. Por ejemplo: la vista Index del controlador OrdersController no requiere ningún parámetro, devuelve una vista con una colección de tipo Order (Compra), solo puede ser accedida por usuarios autenticados entonces la acción requiere el atributo Authorize.

6. Traslado de archivos estáticos:

   En general se trata de archivos CSS, JavaScript, librerías JavaScript e imágenes. Según la estructura de directorios de un proyecto MVC, los archivos CSS deben ir a la carpeta Content al igual que las imágenes. Ahora bien, los archivos y librerías JavaScript se ubicaron en la carpeta Scripts. Estos archivos son los que le dan la apariencia a la aplicación por lo que no se debe olvidar actualizar las referencias a ellos donde sea necesario.

7. Conversión de la Master Page al Layout

Aunque la función de este archivo es la misma en ambos modelos de desarrollo, la de formar una estructura o esqueleto HTML común para todas las interfaces de usuario de la página, se diferencian en la sintaxis que maneja cada una de ellas; por una parte los controles de servidor por la otra el lenguaje Razor.

En el anexo 1 se muestra la estructura del Site Master del ejemplo tratado en este trabajo bajo el modelo de desarrollo Web Forms y en anexo 2, la conversión hecha al formato MVC. A continuación se exponen algunas consideraciones sobre la realización de esta actividad:

- El código del Site Master tiene gran contenido CSS mezclado con HTML, lo ideal seria separarlos para un mejor manejo de los estándares de desarrollo web, pero a raíz que este tema no es el tema central de la investigación se dejará en la forma en que están.

- La primera acción realizada fue referenciar los archivos CSS y JavaScript al proyecto, para que ambos proyectos tuvieran la misma presentación. Para esto se utilizó el método Url.Content de Razor que convierte una dirección relativa en una absoluta.

  ```
  <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
  type="text/css" />
  ```

- El concepto del ContentPlaceHolder no existe en Razor, es remplazado con la función RederSection y para los casos específicos del titulo y el contenido principal de la vista es sustituido por los métodos ViewBag.Title y RenderBody respectivamente.

- Un página en Web Forms, como su nombre lo indica es un formulario, por lo que la página HTML trae siempre la etiqueta form se tenga o no se tenga un formulario real, en el caso MVC solo se utiliza cuando es necesario; por lo tanto esta etiqueta se borra.

- Como en ASP.NET MVC no existen los controles de servidor, todas las atributos runat=server se eliminan.

- Las direcciones de los links se cambian de las URLs físicas utilizadas por las lógicas (Controlador – Acción- Parámetro) que se usan en MVC. Se puede facilitar el trabajo utilizando la función Html.ActionLink como lo muestra el siguiente ejemplo:

  ```
  @Html.ActionLink("About Us", "AboutUs", "Home");
  ```

  Esta función genera un contenido HTML con todo el contenido de un enlace, pero si solo se quiere la dirección se usa Url.Action

- En la aplicación Tailspin Spyworks hay un control de servidor que hace referencia la estado de la sesión llamado LoginView, en MVC se remplazó

con una vista parcial llamada _LogOnPartial y que contiene el siguiente código:

```
@if(Request.IsAuthenticated) {
    @:Sign Out
}
else {
    @:Sign In
}
```

Las vistas parciales se llaman por medio de la función Html.Partial, para este caso es:

```
@Html.Partial("_LogOnPartial")
```

- En el caso del ejemplo, existe un control ListView, trae todas las categorías que se encuentran en la base de datos. Este control se cambió por una vista parcial (Categories/_List) que consulta y presenta esta información. En el punto 8 se podrá ver como convertir el control ListView a código HTML y Razor.

- Como se puede observar en los anexos 1 y 2, aunque no se elimino el contenido CSS para el Layout, el código queda mucho más limpio y entendible que en el antiguo Site Master

8. Desarrollar el proceso de conversión de cada formulario de Web Forms a MVC:

   Siguiendo como guía las figuras 16 y 17 de la identificación de controladores, acciones y vistas, así como también el procedimiento descrito para este punto en la planeación, se procedió a migrar los formularios web al esquema controlador – acción – vista.

   A continuación se describe brevemente por cada acción (clasificadas por controlador) que actividades y cambios se realizaron para hacer la migración. Para el caso de la conversión de las vistas se tuvo en cuenta que: se deben acomodar a la estructura del layout, no existen los controles de servidor por lo que todo atributo runat=server debe ser eliminado, también que cambian las URLs de físicas a lógicas y que toda información que se presenta a través de controles de enlace de datos debe ser cargada desde el controlador o según el caso a través de vistas parciales.

   En los anexos del 3 al 6 se encuentran los ejemplos de como convertir controles de enlace de datos utilizados en este proyecto (ListView, GridView, Repeater, FormView, etc.) a código HTML combinado con Razor.

   Se puede observar en la lista de acciones que estas poseen un término entre paréntesis, por lo que es importante aclarar que significa cada uno:

- V: significa una acción que carga una vista
- POST: la acción que se ejecuta después del envío de un formulario
- *Partial*: representa una vista parcial

HomeController:

- Index (V): Es equivalente a la página Default.aspx. En Web Forms no carga ningún tipo de información en el Page_Load, pero muestra dos controles: el primero un LoginView que identifica si el usuario esta autenticado para mostrarle un mensaje. El segundo web user control un control que a su vez internamente tiene un control Repeater que trae los productos más populares, es decir aquellos mas comprados. El resto de su información es estática.

  Entonces para MVC se construye una acción que presenta la vista, pero no le pasa ningún dato. El control de los productos más populares se reemplaza con la vista parcial Products/_popularItems y el LoginView por el siguiente código:

```
@if (User.Identity.IsAuthenticated) {
        <text>
                Hi @User.Identity.Name. Thanks for coming back.
        </text>
         }else{
        <text>
                Welcome to the store!
        </text>
}
```

- AboutUs (V): Hace las veces de la página About.aspx. Toda la información de esta página es fija, por lo que se crea la acción que carga la vista y el contenido se pasa casi igual a MVC.

- ContactUs (V): Equivale a la página ContactUs.aspx. Esta página no carga ningún dato desde el Page_Load, pero presenta un formulario a para que los clientes realicen comentarios y sugerencias.

  Por consiguiente para convertirlo a Web Forms se crea la acción que llama la vista y en la vista se coloca el método Razor para crear un formulario (Html.BeginForm()) y se pone el mismo contenido pero dejando de lado los controles de servidor y utilizando Razor, por ejemplo no se utiliza:

```
<asp:TextBox ID="TextBoxYourName" runat="server"
Width="500px"></asp:TextBox>
```

Sino que en MVC se usa:

```
@Html.TextBox("TextBoxYourName", "", new { style = "width:500px;" })
```

- ContactUs (POST): El archivo *codebehind* del formulario ContactUs.aspx posee un código que se ejecuta cuando se envía un formulario, este código pasa a ser parte de la acción POST en MVC, solo se cambia la forma de capturar los datos, ya que no se pueden usar los controles de servidor y el tipo de dato no esta definido en el modelo; se debe utilizar la función Request.

CategoriesController:

- _List (Partial): En la Master Page del proyecto de Web Forms se utiliza un ListView para construir un menú con las categorías de los productos. Para el proyecto MVC se utilizó una vista parcial que busca las categorías en la base de datos y las retorna en la vista. La forma en que se convirtió el ListView en código HTML + Razor se encuentra en el Anexo 3.

ProductsController:

- _popularItems (partial): En la vista de Home/Index se deben presentar los productos más populares. En Web Forms se hizo a través de un control de usuario que en su método Page_Load (cargue de la página) va a la base de datos y consulta la información y luego la presenta con la ayuda del control Repeater.

  Para pasar esta funcionalidad a MVC, en la acción que carga la vista se debe colocar la lógica que se encuentra en el método Page_Load, claro esta haciendo algunas modificaciones pero en si es la misma idea. Luego se manda esta información a la vista y se presenta conforme se convirtió el control Repeater en código compatible con MVC (ver anexo 5).

- List (V): En la aplicación Web Forms es semejante al formulario ProductList.aspx, este formulario puede o no recibir un parámetro, si lo recibe muestra los productos de una categoría pero si no lo recibe lista todos los productos registrados mediante el control ListView.

  Para la transformación a MVC en la acción se consultan los productos y se mandan a la vista, quien los representa según el procedimiento para transformar el ListView en código HTML + Razor (anexo 3).

- Details (V): En la aplicación Web Forms corresponde al formulario ProductDetails.aspx, este formulario recibe un parámetro que le indica el identificador del producto. Por medio del control FormView presenta la información del producto y posibilita agregarlo al producto al carrito de compras. Además lista las observaciones (reviews) hechas al producto mediante el uso de un ListView. Desde aquí formulario se puede ir a la vista que crea una nueva observación.

En el proyecto MVC se creó la acción, también con parámetro. Desde la acción se consulta el producto y se envía a la vista, quien lo representa según se transformo el control FormView en código HTML + Razor. (Ver anexo 6). La información de las observaciones del producto se traen por medio de la vista parcial Reviews/_List.

ReviewsController:

- _List (partial): Para la vista Products/Details se requiere mostrar el listado de observaciones hechas a un producto. En Web Forms se lograba esto por medio de un ListView, para MVC se crea una acción que retorna una vista parcial, la acción consulta las observaciones de un producto por medio de su identificador y después las manda a la vista parcial que las representa basándose en el proceso de transformación hecho para los controles ListView (anexo 3).

- Add (V): Su equivalente en el proyecto Web Forms es la página ReviewAdd.aspx, esta página presenta un formulario que le posibilita al usuario el ingreso de una observación a un producto, esta recibe como parámetro el identificador del producto con el que en el Page_Load valida que el producto exista y carga la interfaz. La interfaz esta construida sobre controles de servidor para formularios los que hubo que transformar en sus similares en Razor. A continuación se presenta una tabla de comparación entre los principales controles de servidor Web Forms para formularios y su posible equivalente en Razor

Tabla 5 - Controles de servidor para formularios y Razor

Controles Web Forms	**Razor**
asp:TextBox con la propiedad multiline=true	Html.TextArea()
asp:TextBox	Html.TextBox()
asp:RadioButton	Html.RadioButton()
asp:CheckBoxList	Html.ListBox()
asp:DropDownList	Html.DropDownList()
asp:HiddenField	Html.Hidden()
asp:TextBox con la propiedad texmode=password	Html.Password()

La acción en MVC recibe un parámetro con el que crea un tipo de dato Review que se manda a la vista y con la que se contruye el formulario.

- Add (POST): Este es el POST con el que se crea una observación, se contruye basándose en el método programado en el codebehind del formulario ReviewAdd.aspx, esta acción obtiene del formulario un tipo de dato Review y lo inserta en la base de datos.

ShoppingCartController:

- AddtoCart: Esta acción no representa el cargue de una vista, ni tampoco el envío de datos por método POST, es una acción cuya función es la de agregar un producto al carrito de compras y redireccionar la página al detalle del carrito de compras (ShoppingCart/Index). En Web Forms el formulario es AddToCart.aspx en cuyo método Page_Load esta el código para agregar el producto al carrito y en el que se baso para hacer lo mismo en la acción MVC.

- Index (V): En Web Forms su función la cumple el formulario MyShoppingCart.aspx. Se encarga de mostrarle al usuario el estado de su carrito de compras y le permite actualizarlo. La página en Web Forms utiliza un control de tipo GridView para presentar los datos y en el método Page_Load calcula los subtotales (teniendo presente los precios y las cantidades de los productos del carrito).

  En MVC la acción busca los registros de carrito de compras asociados con la sesión activa, los envía a la vista y esta los presenta conforme a lo analizado para representar un control GridView en MVC (ver anexo 4). Y además por medio de Razor y utilizando variables acumuladoras se calcula el subtotal del carrito.

- Index (POST): Cuando el usuario actualiza su carrito de compras se ejecuta esta acción, la que verifica si se ha eliminado algún elemento o si actualizo alguna cantidad. La lógica la deducimos del método programado para cuando se envía el formulario MyShoppingCart.aspx

- Checkout (V): Su equivalente en Web Forms es la página Checkout.aspx., se ejecuta cuando el usuario da clic en el botón checkout del carrito de compras. Esta es una interfaz de confirmación, le muestra al usuario el estado de su carrito de compras antes de generar la orden. La página utiliza un control GridView para representar los datos. Adicionalmente en un método del codebehind se calcula el total de la compra.

  La acción MVC por lo tanto obtiene los registros de carrito de compras asociados con la sesión activa, los envía a la vista y esta los presenta conforme a lo analizado para representar un control GridView en MVC (ver anexo 4). Y mediante lógica de presentación obtiene la suma de subtotales.

- Checkout (POST): Se ejecuta cuando el usuario confirma la compra, transforma el carrito de compras en una compra realizada (Order). Para la construcción de esta acción se baso en el codebehind del formulario Checkout.aspx en el método que se ejecuta cuando se envía el formulario. Tanto para esta acción como para la que se encarga de cargar la vista es necesario tener el atributo Authorize, pues requiere autenticación.

OrdersController:

- Index (V): Esta acción se encarga de listar todas las compras realizadas por un usuario. En Web Forms sus funciones son realizadas por la página OrderList.aspx que por medio de un GridView le presenta la información al usuario, desde este punto es posible ver el detalle de cada compra.

  En MVC se programa la acción para que consulte en la base de datos las compras realizadas por el usuario autenticado y luego son pasadas a la vista donde son presentadas de acuerdo a la trasformación hecha del control GridView para MVC que se puede observar en el anexo 4. Es importante recordar que tanto esta acción como la acción Details requieren que el usuario esté autenticado por lo que llevan el atributo Authorize.

- Details (V): El formulario correspondiente a esta acción en Web Forms es OrderDetails.aspx que recibe como parámetro el número de la orden. Dicho formulario presenta el detalle de una compra u orden a través de un GridView y además muestra el encabezado (nombre del cliente, hora de registro, hora de entrega, etc.) de la compra por medio de un control FormView.

  Para convertirlo al formato MVC, se crea una acción que busca en la base de datos los detalles de la compra tomando como parámetro el número de la compra y luego los envía a la vista, donde son mostrados teniendo en cuenta la conversión del control GridView a formato HTML + Razor (ver anexo 4). La parte donde se muestra el encabezado de la compra se hace a través de la vista parcial Orders/_OrderHead a la que se manda como parámetro el número de la compra.

- _OrderHead (partial): Esta es una acción que recibe como parámetro el numero de una compra, dato que se usa para buscar en la base de datos el encabezado de la compra y posteriormente pasarlo a la vista parcial teniendo en cuenta la transformación al modelo MVC del control FormView.

AccountController:

ASP.NET MVC nos facilita mucho las cosas en cuanto a autenticación y manejo de usuarios se refiere al implementar por defecto el controlador AccountController encargado de la gestión de usuarios. Las acciones a ser migradas (LogIn, LogIn POST y LogOff) cuyas funciones en el proyecto Web Forms son realizadas por el formulario Login.aspx vienen construidas en el proyecto y se comportan de acuerdo a lo que se espera. Por lo tanto solo fue

necesario acomodar el contenido y el estilo de la vista LogIn para que sea similar a Login.aspx.

9. Evaluar las funcionalidades de interacción con el usuario:

   En el ejemplo tratado en este trabajo la única función de interacción con el usuario que se realiza en el proyecto ASP.NET Web Forms y que no obtiene la misma funcionalidad en MVC es la validación en el cliente de campos en los formularios. En esta situación se decidió como solución utilizar el complemento JQuery.Validate; este complemento es totalmente compatible con el marco de desarrollo MVC, inclusive en el proyecto se añade por defecto la libreria JQuery.

10. Pruebas Funcionales:

    Se realizaron pruebas a todas y cada una de las funcionalidades del proyecto Tailspin Spyworks en MVC, se probo que se ejecutarán correctamente las acciones: que las salidas de correspondieran a los datos entrados, también se probó que los datos ingresados se guardaran correctamente. Se observó también que los datos estuvieran presentados en forma correcta, que no existieran errores en las redirecciones, ni en los links. Adicionalmente se ensayo la seguridad: se intento acceder a sitios no autorizados para un usuario sin autenticar.

    Los resultados de estas pruebas funcionales, que aunque son pruebas muy superficiales fueron exitosos.

- **Implementación**

Aunque este aplicativo no va a pasar por el proceso de implementación debido a que no se trata de una aplicación de ambiente real, ni tampoco se cuenta con el ambiente real para simularlo, es importante tener en cuenta en este proceso los siguientes aspectos para el caso de una aplicación que si lo hiciera:

- La configuración del IIS (Internet Information Services)
- El análisis del ambiente donde se ejecutará: servidor web, servidor de base de datos, servidor de correo, el estado de la red de comunicaciones, etc.
- La realización de pruebas de rendimiento
- El periodo de transición entre la salida de la antigua aplicación y la entrada de la nueva
- La información y capacitación que se dará a los usuarios finales sobre los cambios realizados y su justificación.
- Plan y estrategia de corrección de errores en ambiente real.

3.4.4 Validación

- **¿Qué son pruebas unitarias?**

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. El objetivo principal de las pruebas unitarias es tomar la más pequeña pieza de software en la aplicación, aislarla del resto del código y determinar si se comporta exactamente como es esperado. Cada prueba unitaria se hace por separado antes de integrarse en módulos para probar las interfaces entre los módulos. Las pruebas unitarias han demostrado su valor, debido a que un gran porcentaje de los defectos son identificados durante su ejecución. (MSDN Library, 2012)

Las pruebas unitarias poseen las siguientes características:

- Automatizable: generalmente no requiere una intervención manual.
- Completas: deben cubrir la mayor cantidad de código.
- Repetibles o Reutilizables
- Independientes: la ejecución de una prueba no debe afectar a la ejecución de otra. (Dos Ideas, 2012)

- **Procedimiento de validación**

Para validar el correcto funcionamiento de la página migrada al modelo de desarrollo ASP.NET MVC se utilizaron pruebas unitarias, aprovechando que el mismo marco de desarrollo facilita el uso de las mismas, ya que permite el desarrollo guiado por pruebas (TDD).

El procedimiento se realizó comparando los datos a retornar de cada acción en los controladores y compararlos con los resultados esperados, teniendo en cuenta que los resultados esperados son aquellos que muestra la aplicación desarrollada en Web Forms, por lo que se construyeron los valores esperados a partir de los datos que se observaron en ese momento especifico en la aplicación Web Forms.

Por ejemplo la acción Products/Details/357 se debe obtener como resultado un producto cuyo nombre es “Escape Vehicle (Air)” y a continuación se muestra el código de la prueba unitaria realizada:

```
[TestMethod]
        public void Details()
        {
            ProductsController target = new ProductsController();
            int id = 357; //
            string expected = "Escape Vehicle (Air)”";
            ViewResult actual;
            actual = target.Details(id) as ViewResult;

            Product miProducto = actual.ViewData.Model as Product;

            Assert.AreEqual(expected, miProducto.ModelName);
            Assert.Inconclusive("Verify the correctness of this test method.");
        }
```

El resultado de la anterior prueba fue exitoso, con lo que se puede ver que se migró correctamente la acción de Web Forms a MVC. Para ver las demás pruebas unitarias favor revisar el proyecto TailspinSpyworks que se encuentra como archivo digital adjunto a este trabajo de grado.

4. RESULTADOS OBTENIDOS

El propósito principal desde la planeación de este proyecto era presentar un procedimiento que ayudara a la migración de una aplicación de ASP.NET Web Forms a ASP.NET MVC, para lo cual inicialmente se realizó una investigación sobre el modelo de desarrollo ASP.NET Web Forms, la cual se puede ver en la sección 3.1 de este trabajo y en donde se describen de manera general las principales características de este modelo de desarrollo incluyendo el patrón Page Controller bajo el cual trabaja, la presentación de datos, el ciclo de vida de la página. De esta es importante destacar el importante papel que juegan los mecanismos postback y view state, así como también los controles de servidor como herramienta de ayuda para la presentación de datos.

Luego se pasó a construir la exploración del modelo ASP.NET MVC (sección 3.2), en la que se observa primero una descripción general del modelo de desarrollo, la estructura del proyecto, el ciclo de vida de la página y los patrones de diseño que lo soportan: MVC, Front Controller y Model2. Después se encuentra la explican cada uno de las tres responsabilidades en que se separa el modelo de desarrollo: el modelo, la vista y el controlador. Además se identifican las características de seguridad provistas por MVC, como lo son la autorización y la prevención de ataques XSS y CSRF.

Posteriormente y con base en las investigaciones realizadas a Web Forms y a MVC, se hizo un análisis de las principales similitudes y diferencias de cada uno, se elaboró un paralelo y se enumeraron las ventajas y desventajas de cada modelo de desarrollo. Es de vital relevancia decir que no hay un modelo de desarrollo mejor o superior que el otro, todo depende del tipo de proyecto a realizar y de las necesidades especificas que se tengan. Este análisis se puede ver con más detalle en la sección 3.3.

Después se desarrollo en procedimiento de migración, punto central de este proyecto, para lograrlo se empezó con buscar ejemplos de metodologías utilizadas por empresas especializadas en asesorar este tipo de procesos, para apoyándose en dichas metodologías definir una para este trabajo. La metodología consistió en cuatro pasos: análisis, planeación, desarrollo e implementación; a la medida que se explica cada uno de los pasos se realizó el ejemplo documentado, el código del ejemplo de migración se encuentra como archivo digital adjunto a este trabajo. El procedimiento realizado fue exitoso se logró migrar a MVC una aplicación llamada Tailspin Spyworks, que básicamente se trata de una tienda en línea, se tuvieron problemas especialmente en el rediseño de las interfaces gráficas pero realizando un análisis del código HTML generado por la aplicación Web Forms y teniendo en cuenta el aspecto final de la pagina se pudo llegar a buen termino.

Finalmente se realizó el procedimiento de validación por medio de pruebas unitarias, utilizando las herramientas que ofrece el ambiente de desarrollo que facilitan la construcción de estas, donde se probó que las características funcionales del proyecto Web Forms se conservarán en MVC, esto probando principalmente que los datos de salida de las acciones coincidieran con lo mostrado por la aplicación Web Forms, obteniendo resultados satisfactorios.

5. CONCLUSIONES Y CONSIDERACIONES

- ASP.NET Web Forms, es una muy buena herramienta, por algo se ha mantenido popular por tantos años y se han construido tantas aplicaciones con tan buenos resultados. Lo malo es que muchos desarrolladores hacen un mal uso de ella, más por desconocimiento que por otra cosa, aplicando malas prácticas de desarrollo que la sobrecargan y de ahí viene su mala reputación.

- ASP.NET MVC es el futuro del desarrollo web bajo el marco de trabajo .NET, se ha aprovechado de las limitantes de su antecesor para crear un modelo de desarrollo impresionante de acuerdo a las necesidades actuales del desarrollo para la web. A pesar de llevar relativamente poco tiempo ha tomado mucha fuerza, ya posee gran cantidad de elementos compatibles y ha logrado que muchos se interesen en aprender más sobre el tema.

- Tanto ASP.NET Web Forms como ASP.NET MVC poseen cada uno características muy buenas. Definitivamente no se puede decir que una sea mejor que la otra, todo depende de las necesidades de la aplicación por desarrollar, del tiempo de desarrollo, de los recursos disponibles, de la disponibilidad de hosting y de la experiencia del equipo desarrollo.

- Se encuentra muy poca información acerca de como convertir una aplicación de Web Forms a MVC, en algunos aspectos el proceso se vuelve algo engorroso debido a que algunas funcionalidades es necesario rediseñarlas de nuevo, por ejemplo el caso de las interfaces gráficas donde muchas veces es mejor construir una nueva interfaz para no ir en contra de los estándares HTML.

- Antes de realizar un proceso similar al realizado en este trabajo, es bueno evaluar detalladamente las razones por las que se realiza, en términos de tiempos de desarrollo, inversión económica, rendimiento, funcionalidad, etc. Para conocer si se justifica el proceso o si por el contrario es mejor dejarla en el estado en que está.

- Se recomienda para futuros trabajos sobre el tema, dividir el estudio en aspectos más específicos; cualquier aplicación web involucra muchos temas: acceso a datos, lógica de negocio, lógica de presentación, seguridad, pruebas, etc. Si el caso práctico analizado en este trabajo se puede considerar relativamente simple y dio para tanto contenido, como lo será una con un estado de madurez alto y de un ambiente real.

- Se espera que el procedimiento realizado en este documento sirva como información base para futuros proyectos, ya que según lo investigado se encuentra muy poca información relativa a como migrar una aplicación construida en ASP.NET Web Forms a ASP.NET MVC. Es necesario por lo tanto el aporte de nuevas ideas y alternativas a las planteadas en este trabajo para darle una mayor madurez al procedimiento.

BIBLIOGRAFÍA

Aguilar, J. M. (03 de 05 de 2010). *Variable not found*. Recuperado el 21 de 02 de 2011, de Variable not found: http://www.variablenotfound.com/2010/05/aspnet-mvc-2-quince-cuestiones-que.html#p21

Alameda, V. (04 de 12 de 2009). *Victor Alameda*. Recuperado el 07 de 03 de 2011, de Victor Alameda: http://developersdotnet.com/blogs/valameda/archive/2009/04/12/191-qu-233-es-asp-net-mvc.aspx

ASP.NET. (17 de 10 de 2012). *ASP.NET MVC Overview.* Recuperado el 17 de 10 de 2012, de ASP.NET MVC Overview.

ASP.NET. (14 de 10 de 2012). *What is Web Forms?* Recuperado el 14 de 10 de 2012, de What is Web Forms?: http://www.asp.net/web-forms/what-is-web-forms

Dos Ideas. (21 de 10 de 2012). *Prueba Unitaria.* Recuperado el 21 de 10 de 2012, de Prueba Unitaria: http://www.dosideas.com/wiki/Prueba_Unitaria

Esposito, D. (12 de 2008). *Cutting Edge: ASP.NET Presentation Patterns - MSDN Magazine.* Recuperado el 19 de 10 de 2012, de Cutting Edge: ASP.NET Presentation Patterns - MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/dd252940.aspx

Esposito, D. (2010). *Programming Microsoft ASP.NET MVC.* Redmond, Washington: Microsoft Press.

Garcia Puebla, I. (27 de 01 de 2008). *Eventos de ciclo de vida de ASP.NET.* Recuperado el 14 de 10 de 2012, de Eventos de ciclo de vida de ASP.NET: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=eventosASPNET

Guthrie, S. (02 de 07 de 2010). *Introducing "Razor" – a new view engine for ASP.NET.* Recuperado el 19 de 10 de 2012, de Introducing "Razor" – a new view engine for ASP.NET: http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx

Inteq. (21 de 10 de 2012). *Application Migration Services for Toxicology Labs.* Recuperado el 21 de 10 de 2012, de Application Migration Services for Toxicology Labs: http://www.inteqsolutions.com/html/toxicology_labs.htm#

IT Consultancy Services. (20 de 10 de 2012). *Application Migration.* Recuperado el 20 de 10 de 2012, de Application Migration: http://www.itcservices.in/application-migration.html

López Ramírez, J. (10 de 11 de 2009). *La historia de mis desventuras, Palabras más, palabras menos sobre desarrollo de software*. Recuperado el 20 de 02 de 2011, de La historia de mis desventuras, Palabras más, palabras menos sobre desarrollo de software: http://calamitatum.wordpress.com/tag/webforms/

Malcolm, G. (15 de 10 de 2012). *User Interface Patterns - geekswithblogs.* Recuperado el 15 de 10 de 2012, de User Interface Patterns - geekswithblogs: http://geekswithblogs.net/gregorymalcolm/archive/2009/07/14/user-interface-patterns.aspx

Menendez Soto, S., & Robledo Fong, O. (2002). Migración de Sistemas. *Migración de Sistemas*. Guatemala: Universidad Francisco Marroquin.

Mikesdotnetting. (11 de 05 de 2009). *ASP.NET MVC Partial Views and Strongly Typed Custom ViewModels.* Recuperado el 19 de 10 de 2012, de ASP.NET MVC Partial Views and Strongly Typed Custom ViewModels: http://www.mikesdotnetting.com/Article/105/ASP.NET-MVC-Partial-Views-and-Strongly-Typed-Custom-ViewModels

MSDN. (14 de 10 de 2012). *¿Qué es el desarrollo web? (información general).* Recuperado el 14 de 10 de 2012, de ¿Qué es el desarrollo web? (información general): http://msdn.microsoft.com/es-ES/ff380144#one

MSDN Library. (2011). *ASP.NET Page Life Cycle Overview.* Recuperado el 25 de 05 de 2011, de ASP.NET Page Life Cycle Overview: http://msdn.microsoft.com/en-us/library/ms178472.aspx

MSDN Library. (04 de 03 de 2012). *ASP.NET and Visual Web Developer.* Recuperado el 04 de 03 de 2012, de ASP.NET and Visual Web Developer: http://msdn.microsoft.com/en-us/library/dd566231.aspx

MSDN Library. (04 de 03 de 2012). *ASP.NET Architecture.* Recuperado el 04 de 03 de 2012, de ASP.NET Architecture: http://msdn.microsoft.com/en-us/library/aa719552(v=VS.71).aspx

MSDN Library. (19 de 10 de 2012). *ASP.NET Configuration.* Recuperado el 19 de 10 de 2012, de ASP.NET Configuration: http://msdn.microsoft.com/en-us/library/aa719558.aspx

MSDN Library. (15 de 10 de 2012). *ASP.NET Data Access Overview.* Recuperado el 15 de 10 de 2012, de ASP.NET Data Access Overview: http://msdn.microsoft.com/en-us/library/ms178359(v=vs.100).aspx

MSDN Library. (20 de 10 de 2012). *ASP.NET Master Pages.* Recuperado el 20 de 10 de 2012, de ASP.NET Master Pages: http://msdn.microsoft.com/en-us/library/wtxbf3hh(v=vs.100).aspx

MSDN Library. (19 de 10 de 2012). *ASP.NET MVC Overview.* Recuperado el 19 de 10 de 2012, de ASP.NET MVC Overview: http://msdn.microsoft.com/en-us/library/dd381412(VS.98).aspx

MSDN Library. (16 de 10 de 2012). *ASP.NET Routing.* Recuperado el 16 de 10 de 2012, de ASP.NET Routing: http://msdn.microsoft.com/en-us/library/cc668201.aspx#routes

MSDN Library. (19 de 10 de 2012). *ASP.NET Web Server Controls Overview.* Recuperado el 19 de 10 de 2012, de ASP.NET Web Server Controls Overview: http://msdn.microsoft.com/en-us/library/zsyt68f1.aspx

MSDN Library. (19 de 10 de 2012). *Controllers and Action Methods in ASP.NET MVC Applications.* Recuperado el 19 de 10 de 2012, de Controllers and Action Methods in ASP.NET MVC Applications: http://msdn.microsoft.com/en-us/library/dd410269(VS.98).aspx

MSDN Library. (15 de 10 de 2012). *Data Source Controls Overview.* Recuperado el 15 de 10 de 2012, de Data Source Controls Overview: http://msdn.microsoft.com/en-us/library/ms227679(v=vs.100).aspx

MSDN Library. (14 de 10 de 2012). *Introduction to ASP.NET and Web Forms .* Recuperado el 14 de 10 de 2012, de Introduction to ASP.NET and Web Forms : http://msdn.microsoft.com/en-us/library/ms973868.aspx

MSDN Library. (20 de 10 de 2012). *MVC Framework and Application Structure.* Recuperado el 20 de 10 de 2012, de MVC Framework and Application Structure: http://msdn.microsoft.com/en-us/library/dd410120(v=vs.98).aspx

MSDN Library. (04 de 09 de 2012). *Overview of .NET Framework.* Recuperado el 04 de 09 de 2012, de Overview of .NET Framework: http://msdn.microsoft.com/library/zw4w595w.aspx

MSDN Library. (18 de 10 de 2012). *PostBacks and round trips.* Recuperado el 18 de 10 de 2012, de PostBacks and round trips: http://msdn.microsoft.com/en-us/library/ms178125.aspx

MSDN Library. (17 de 10 de 2012). *Understanding MVC Application Execution.* Recuperado el 17 de 10 de 2012, de Understanding MVC Application Execution: http://msdn.microsoft.com/en-us/library/dd381612(v=vs.98).aspx

MSDN Library. (21 de 10 de 2012). *Unit Testing.* Recuperado el 21 de 10 de 2012, de Unit Testing: http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

Rapidsoft Systems. (20 de 10 de 2012). *Application Migration and Redeployment Services.* Recuperado el 20 de 10 de 2012, de Application Migration and Redeployment Services: http://www.rapidsoftsystems.com/application-migration-services-L2.html

Sanderson, S. (2010). *Pro ASP.NET MVC 2 Framework.* Apress.

Talend. (2010). *Talen Open Data Solutions*. Recuperado el 07 de 03 de 2011, de Talen Open Data Solutions: http://es.talend.com/solutions-data-integration/data-migration.php

Thai, T., & Lam, H. (2002). *.NET Framework Essentials.* o'reilly.

Trowbridge, D., Mancini, D., Quick, D., Hohpe, G., Newkirk, J., & Lavigne, D. (14 de 10 de 2012). *Microsoft patterns & practices.* Recuperado el 14 de 10 de 2012, de Microsoft patterns & practices: http://msdn.microsoft.com/en-us/library/ff650511.aspx

Venulopagan, N. (14 de 10 de 2012). *An Overview of Authentication and Authorization Options in ASP.NET.* Recuperado el 14 de 10 de 2012, de An Overview of Authentication and Authorization Options in ASP.NET: http://www.4guysfromrolla.com/articles/031204-1.aspx

Wikipedia. (07 de 03 de 2011). *Wikipedia*. Recuperado el 07 de 03 de 2011, de Wikipedia: http://es.wikipedia.org/wiki/Migraci%C3%B3n

Wikipedia. (06 de 10 de 2012). *ASP.NET.* Recuperado el 06 de 10 de 2012, de ASP.NET: http://en.wikipedia.org/wiki/ASP.NET

Wikipedia. (04 de 03 de 2012). *Common Language Runtime.* Recuperado el 04 de 03 de 2012, de Common Language Runtime: http://es.wikipedia.org/wiki/Common_Language_Runtime

Wikipedia. (20 de 10 de 2012). *Cross Site Request Forgery.* Recuperado el 20 de 10 de 2012, de Cross Site Request Forgery: http://es.wikipedia.org/wiki/Cross_Site_Request_Forgery

Wikipedia. (04 de 03 de 2012). *Microsoft .NET.* Recuperado el 04 de 03 de 2012, de Microsoft .NET: http://es.wikipedia.org/wiki/Microsoft_.NET

Wiley Online Library. (03 de 10 de 2012). *Wiley Online Library.* Recuperado el 03 de 10 de 2012, de Wiley Online Library: http://media.wiley.com/product_data/excerpt/98/07645482/0764548298.pdf

ANEXO 1: MASTER PAGE EN WEB FORMS

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs"
Inherits="TailspinSpyworks.SiteMaster" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head runat="server">
    <title></title>
    <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />

<asp:ContentPlaceHolder ID="HeadContent" runat="server"></asp:ContentPlaceHolder>
</head>

<body>
<form runat="server" id="MyForm">
 <div id="outer">
       <div id="header">
              <div style="float: left;">
                     <h1><a href="#">Tailspin Spyworks [Web Forms]</a></h1>
                     <h2>Powered by ASP.NET</h2>
              </div>
              <div style="float: right; vertical-align:middle;">
                     <br /><br />
                     <table cellpadding="0" cellspacing="0" border="0">
                            <tr style="vertical-align:middle;" >

                            <td align="center" style="width: 65px">
                            <a id="A1" href="~/Account/Login.aspx" runat="server"
                            class="SiteLinkBold">
                            <img id="Img1" src="~/Styles/Images/sign_in.gif" alt=""
                            runat="server" /><br />

                            <asp:LoginView ID="MenuLoginView" runat="server">
                                   <AnonymousTemplate>
                                          Sign In
                                   </AnonymousTemplate>
                                   <LoggedInTemplate>
                                          Log Out
                                   </LoggedInTemplate>
                            </asp:LoginView>
                            </a>
                            </td>

                            <td align="center" style="width: 75">
                            <a id="A2" href="~/Account/OrderList.aspx" runat="server"
                            class="SiteLinkBold">
                            <img id="Img2" src="~/Styles/Images/account.gif" alt=""
                            runat="server" /><br />Account
                            </a>
```

```
                        </td>

                        <td align="center" style="width: 65"><a id="A3"
                        href="~/MyShoppingCart.aspx"  runat="server"
                        class="SiteLinkBold">
                        <img id="Img3" src="~/Styles/Images/cart.gif" alt=""
                        runat="server" /><br />Cart
                        </a>
                        </td>

                        <td align="center" style="width:
                        65">         
                        ; 
                        </td>

                        </tr>
                </table>
        </div>
</div>

<div id="subheader">
        <a id="A4" href="~/Default.aspx" runat="server" title="">Home</a>
        <a id="A5" href="~/ProductsList.aspx" runat="server"
        title="">Products</a>
        <a id="A6" href="~/AboutUs.aspx" runat="server" title="">About Us</a>
        <a id="A7" href="~/ContactUs.aspx" runat="server" title="">Contact
        Us</a>
</div>

<div id="content">
        <div id="rightColumn">
        </div>
        <div id="mainContent">
                <div id="centerColumn">
                        <asp:ContentPlaceHolder ID="MainContent" runat="server">
                                <p>
                                </p>
                        </asp:ContentPlaceHolder>
                </div>
        </div>

        <div id="leftColumn">
                <h3>Our Products</h3>

                <asp:ListView ID="ListView_CategoryMenu" runat="server"
                DataKeyNames="CategoryID"
                DataSourceID="EDS_Category_Menu">

                <EmptyDataTemplate>No Menu Items.</EmptyDataTemplate>
                <ItemSeparatorTemplate></ItemSeparatorTemplate>
                <ItemTemplate>

                <li>
```

```
                        <a href='<%#
                        VirtualPathUtility.ToAbsolute("~/ProductsList.aspx?CategoryId
                        ="+Eval("CategoryID")) %>'><%# Eval("CategoryName") %></a>
                    </li>
                    </ItemTemplate>
                    <LayoutTemplate>

                    <ul ID="itemPlaceholderContainer" runat="server" style="font-
                    family:Verdana, Arial, Helvetica, Sans-Serif;">
                        <li runat="server" id="itemPlaceholder" />
                    </ul>

                    <div style="text-align:center; background-color:#FFCC66; font-
                    family:Verdana, Arial, Helvetica, Sans-Serif; color:#333333;">
                    </div>

                    </LayoutTemplate>
                    </asp:ListView>

                    <asp:EntityDataSource ID="EDS_Category_Menu" runat="server"
                    ConnectionString="name=CommerceEntities"
                    DefaultContainerName="CommerceEntities" EnableFlattening="False"
                    EntitySetName="Categories">
                    </asp:EntityDataSource>

                </div>
                <div class="clear"></div>
            </div>
            <div id="footer"><p>Copyright &copy; 2010 Tailspin Spyworks</p></div>
</div>

</form>
</body>
</html>
```

ANEXO 2: LAYOUT ASP.NET MVC

```
<!DOCTYPE html>
<html>
<head>
       <title>@ViewBag.Title</title>
       <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet"
       type="text/css"   />
       <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"
       type="text/javascript"></script>
</head>

<body>
    <div id="outer">
           <div id="header">
                  <div style="float: left;">
                         <h1><a href="@Url.Action("Index","Home")">Tailspin Spyworks
                         [MVC]</a></h1>
                         <h2>Powered by ASP.NET</h2>
                  </div>
                  <div style="float: right; vertical-align:middle;">
                      <br /><br />

                <table cellpadding="0" cellspacing="0" border="0">
                    <tr style="vertical-align:middle;" >
                           <td align="center" style="width: 65px">
                            <a id="A1" href="@Url.Action("LogIn", "Account")"
                           class="SiteLinkBold">
                           <img id="Img1"
                           src="../../Content/themes/base/images/sign_in.gif" alt=""
                           /><br />
                                @Html.Partial("_LogOnPartial")
                            </a>
                           </td>

                           <td align="center" style="width: 75">
                            <a id="A2" href="@Url.Action("Index","Orders")"
                           class="SiteLinkBold">
                           <img id="Img2"
                           src="../../Content/themes/base/images/account.gif" alt=""
                           />
                           <br />Account
                           </a>
                           </td>

                           <td align="center" style="width: 65">
                           <a id="A3" href="@Url.Action("Index", "ShoppingCart")"
                           class="SiteLinkBold">
                           <img id="Img3"
                           src="../../Content/themes/base/images/cart.gif" alt="" />
                           <br />Cart
                           </a>
```

```
                        </td>

                        <td align="center" style="width: 65">
                                  
                        </td>
                    </tr>
                </table>
                </div>
        </div>

        <div id="subheader">
                @Html.ActionLink("Home", "Index", "Home")
                @Html.ActionLink("Products", "/../Products/List", new { id = 0 })
                @Html.ActionLink("About Us", "AboutUs", "Home");
                @Html.ActionLink("Contact Us", "ContactUs", "Home" );

        </div>

        <div id="content">
                <div id="rightColumn">
                </div>
                <div id="mainContent">
                        <div id="centerColumn">
                                @RenderBody()
                        </div>
                </div>
                <div id="leftColumn">
                        <h3>Our Products</h3>
                        @Html.Action("_List", "Categories")
                </div>
                <div class="clear"></div>
         </div>
         <div id="footer"><p>Copyright &copy; 2010 Tailspin Spyworks</p></div>
    </div>

</body>
</html>
```

ANEXO 3: EJEMPLO DE CONVERSIÓN CONTROL LISTVIEW

Código del control ListView

```
<asp:ListView ID="ListView_CategoryMenu" runat="server" DataKeyNames="CategoryID"
DataSourceID="EDS_Category_Menu">
       <EmptyDataTemplate>No Menu Items.</EmptyDataTemplate>
       <ItemSeparatorTemplate></ItemSeparatorTemplate>
       <ItemTemplate>
              <li>
                     <a href='<%#
                     VirtualPathUtility.ToAbsolute("~/ProductsList.aspx?CategoryId="+
                     Eval("CategoryID")) %>'><%# Eval("CategoryName") %></a>
              </li>
       </ItemTemplate>
       <LayoutTemplate>
              <ul ID="itemPlaceholderContainer" runat="server" style="font-
              family:Verdana, Arial, Helvetica, Sans-Serif;">
                     <li runat="server" id="itemPlaceholder" />
              </ul>

              <div style="text-align:center; background-color:#FFCC66; font-
              family:Verdana, Arial, Helvetica, Sans-Serif; color:#333333;">
              </div>
       </LayoutTemplate>
 </asp:ListView>
```

Código HTML generado por el control:

```
<ul id="ListView_CategoryMenu_itemPlaceholderContainer" style="font-
family:Verdana, Arial, Helvetica, Sans-Serif;">

        <li>
        <a href='/ProductsList.aspx?CategoryId=14'>Communications</a>
       </li>

       <li>
       <a href='/ProductsList.aspx?CategoryId=15'>Deception</a>
       </li>

       <li>
       <a href='/ProductsList.aspx?CategoryId=16'>Travel</a>
       </li>

       <li>
       <a href='/ProductsList.aspx?CategoryId=17'>Protection</a>
       </li>
```

```
        <li>
        <a href='/ProductsList.aspx?CategoryId=18'>Munitions</a>
        </li>

        <li>
        <a href='/ProductsList.aspx?CategoryId=19'>Tools</a>
        </li>

        <li>
        <a href='/ProductsList.aspx?CategoryId=20'>General</a>
        </li>
</ul>

<div style="text-align:center; background-color:#FFCC66; font-
family:Verdana, Arial, Helvetica, Sans-Serif; color:#333333;">
</div>
```

De acuerdo al control ListView y al código HTML generado se dedujo el siguiente código para las vistas del proyecto MVC

```
<ul style="font-family:Verdana, Arial, Helvetica, Sans-Serif;">

    @foreach (var item in Model) {
        <li>

         @Html.ActionLink(item.CategoryName, "/../Products/List", new { id =
       item.CategoryID })

        </li>
    }
</ul>

<div style="text-align:center; background-color:#FFCC66; font-family:Verdana, Arial,
Helvetica, Sans-Serif; color:#333333;">
</div>
```

ANEXO 4: EJEMPLO DE CONVERSIÓN CONTROL GRIDVIEW

Código del control GridView

```
<asp:GridView ID="MyList" runat="server" AutoGenerateColumns="False"
ShowFooter="true"
            GridLines="Vertical" CellPadding="4"
            DataSourceID="EDS_Cart"
            DataKeyNames="ProductID, UnitCost, Quantity"
            CssClass="CartListItem">

        <AlternatingRowStyle CssClass="CartListItemAlt"/>
        <Columns>
             <asp:BoundField DataField="ProductID" HeaderText="Product ID"
             ReadOnly="True" SortExpression="ProductID" />
            <asp:BoundField DataField="ModelNumber" HeaderText="Model Number"
            ReadOnly="True" SortExpression="ModelNumber" />
            <asp:BoundField DataField="ModelName" HeaderText="Model Name"
            ReadOnly="True" SortExpression="ModelName" />
            <asp:BoundField DataField="UnitCost" HeaderText="Unit Cost"
            ReadOnly="True" SortExpression="UnitCost" DataFormatString="{0:c}" />

            <asp:TemplateField>
                <HeaderTemplate>Quantity</HeaderTemplate>
                <ItemTemplate>
                    <asp:TextBox ID="PurchaseQuantity" Width="40" runat="server"
                            Text='<%# Bind("Quantity") %>'></asp:TextBox>
                </ItemTemplate>
            </asp:TemplateField>

            <asp:TemplateField>
                <HeaderTemplate>Item Total</HeaderTemplate>
                <ItemTemplate>
                    <%# (Convert.ToDouble(Eval("Quantity")) *
                    Convert.ToDouble(Eval("UnitCost"))) %>
                </ItemTemplate>
            </asp:TemplateField>

            <asp:TemplateField>
                <HeaderTemplate>Remove Item</HeaderTemplate>
                <ItemTemplate>
                    <center>
                        <asp:CheckBox id="Remove" runat="server" />
                    </center>
                </ItemTemplate>
            </asp:TemplateField>
        </Columns>
        <FooterStyle CssClass="CartListFooter" />
        <HeaderStyle CssClass="CartListHead" />
    </asp:GridView>
```

Código HTML generado por el GridView

```
<table class="CartListItem" cellspacing="0" cellpadding="4" rules="cols"
border="1" id="MainContent_MyList" style="border-collapse:collapse;">
        <tr class="CartListHead">
                <th scope="col">Product ID</th>
                <th scope="col">Model Number</th>
                <th scope="col">Model Name</th>
                <th scope="col">Unit Cost</th>
                <th scope="col">Quantity</th>
                <th scope="col">Item Total</th>
                <th scope="col">Remove Item</th>
        </tr>
        <tr>
                <td>377</td>
                <td>BME007</td>
                <td>Identity Confusion Device</td>
                <td>$ 6,99</td>
                <td>
                    <input
                  name="ctl00$MainContent$MyList$ctl02$PurchaseQuantity"
                  type="text" value="1"
                  id="MainContent_MyList_PurchaseQuantity_0"
                  style="width:40px;" />
                 </td>
                <td>6.99</td>
                <td>
                        <center>
                        <input id="MainContent_MyList_Remove_0"
                        type="checkbox"
                        name="ctl00$MainContent$MyList$ctl02$Remove" />
                        </center>
                 </td>
        </tr>
        <tr class="CartListItemAlt">
                <td>387</td>
                <td>SQRTME1</td>
                <td>Remote Foliage Feeder</td>
                <td>$ 9,99</td>
                <td>
                        <input
                        name="ctl00$MainContent$MyList$ctl03$PurchaseQuant
                        ity" type="text" value="1"
                        id="MainContent_MyList_PurchaseQuantity_1"
                        style="width:40px;" />
                 </td>
                <td>9.99</td>
                <td>
                     <center>
                        <input id="MainContent_MyList_Remove_1"
                        type="checkbox"
                        name="ctl00$MainContent$MyList$ctl03$Remove" />
                     </center>
```

```
                </td>
                </tr>
                <tr class="CartListFooter">
                        <td> </td>
                        <td> </td>
                        <td> </td>
                        <td> </td>
                        <td> </td>
                        <td> </td>
                        <td> </td>
                </tr>
        </table>
```

De acuerdo al control GridView y al código HTML generado se dedujo el siguiente código para las vistas del proyecto MVC

```
<table id="MainContent_MyList" class="CartListItem"  cellspacing="0" cellpadding="4"
border="1" style="border-collapse:collapse;" rules="cols">
            <tr class="CartListHead">
                <th scope="col">Product ID</th>
                <th scope="col">Model Number</th>
                <th scope="col">Model Name</th>
                <th scope="col">Unit Cost</th>
                <th scope="col">Quantity</th>
                <th scope="col">Item Total</th>
                <th scope="col">Remove Item</th>
            </tr>
            @foreach (var item in Model)
            {
                costo = item.Product.UnitCost * item.Quantity;
                total = total + costo;
                cont++;
                if (cont % 2 == 1)
                {
                    <tr>
                        <td>@item.ProductID</td>
                        <td>@item.Product.ModelNumber</td>
                        <td>@item.Product.ModelName</td>

                        <td>@(String.Format("{0:c}", item.Product.UnitCost))</td>
                         <td>@Html.TextBox("PurchaseQuantity" + item.ProductID,
                    item.Quantity, new { style = "width:40px;" })
                        </td>

                        <td>@(String.Format("{0:c}", costo))</td>
                        <td>
                            <center>
                                @Html.CheckBox("remove" + item.ProductID)
                            </center>
                        </td>
                    </tr>
                }
                else
                {
```

```
                <tr class="CartListItemAlt">
                    <td>@item.ProductID</td>
                    <td>@item.Product.ModelNumber</td>
                    <td>@item.Product.ModelName</td>

                    <td>@(String.Format("{0:c}", item.Product.UnitCost))</td>

                       <td>@Html.TextBox("PurchaseQuantity" + item.ProductID,
                       item.Quantity, new { style = "width:40px;" })
                     </td>



                    <td>@(String.Format("{0:c}", costo))</td>
                    <td>
                        <center>
                            @Html.CheckBox("remove" + item.ProductID)
                        </center>
                    </td>
                </tr>
            }

        }
        <tr class="CartListFooter">
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
    </table>
```

ANEXO 5: EJEMPLO DE CONVERSIÓN CONTROL REPEATER

Código del control Repeater

```
<asp:Repeater ID="RepeaterItemsList" runat="server">
            <HeaderTemplate></HeaderTemplate>
            <ItemTemplate>
                    <a class='MostPopularItemText'
                    href='ProductDetails.aspx?productID=<%# Eval("ProductId") %>'>
                    <%# Eval("ModelName") %></a>
                    <br />
            </ItemTemplate>
            <FooterTemplate></FooterTemplate>
 </asp:Repeater>
```

Código HTML generado por el control

```
<a class='MostPopularItemText' href='ProductDetails.aspx?productID=355'>
Rain Racer 2000</a>
<br />

<a class='MostPopularItemText' href='ProductDetails.aspx?productID=371'>
Mighty Mighty Pen</a>
<br />

<a class='MostPopularItemText' href='ProductDetails.aspx?productID=390'>
Rubber Stamp Beacon</a>
<br />

<a class='MostPopularItemText' href='ProductDetails.aspx?productID=378'>
Guard Dog Pacifier</a>
<br />

<a class='MostPopularItemText' href='ProductDetails.aspx?productID=397'>
Nonexplosive Cigar</a>
<br />
```

De acuerdo al control Repeater y al código HTML generado se dedujo el siguiente código para las vistas del proyecto MVC

```
@foreach (var item in Model) {
        @Html.ActionLink(item.ModelName, "Details", "Products", new { id =
        item.ProductID }, new { @class = "MostPopularItemText" })
        <br />
}
```

ANEXO 6: EJEMPLO DE CONVERSIÓN CONTROL FORMVIEW

Código del control del FormView

```
<asp:FormView ID="FormView_Product" runat="server"
DataKeyNames="ProductID" DataSourceID="EDS_Product">

       <ItemTemplate>
              <div class="ContentHead"><%# Eval("ModelName") %></div><br />
             <table border="0">
                <tr>
                    <td style="vertical-align:top;">
                        <img src='Catalog/Images/<%# Eval("ProductImage") %>'
                        border="0" alt='<%# Eval("ModelName") %>'/>
                    </td>
                    <td style="vertical-align:top;">
                        <%# Eval("Description") %>
                        <br /><br /><br />
                    </td>
                </tr>
             </table>

             <span class="UnitCost">
                <b>Your Price:</b> <%# Eval("UnitCost", "{0:c}") %>
                <br />
                <span class="ModelNumber">
                <b>Model Number</b> <%# Eval("ModelNumber") %>
                </span>
                <br />

                <a href='AddToCart.aspx?ProductID=<%# Eval("ProductID") %>'
                style="border:none, 0, white">
                    <img id="Img1" src="~/Styles/Images/add_to_cart.gif"
                    runat="server"
                    alt="" style="border-width:0"/>
                </a>
                <br /><br />
            </span>

            <div class="ContentHead">Review</div><br />
             <a id="ReviewList_AddReview" href="ReviewAdd.aspx?productID=<%#
             Eval("ProductID") %>">
                <img id="Img2" runat="server" style="vertical-align:bottom;"
                    src="~/Styles/Images/review_this_product.gif" alt=""/>
             </a>
        </ItemTemplate>

    </asp:FormView>
```

Código HTML generado por el control

```
<table cellspacing="0" id="MainContent_FormView_Product" style="border-
collapse:collapse;">
        <tr>

        <td colspan="2">
             <div class="ContentHead">Mighty Mighty Pen</div><br />

             <table border="0">
                <tr>
                    <td style="vertical-align:top;">
                    <img src='Catalog/Images/image.gif'
                    border="0" alt='Mighty Mighty Pen'/>
                    </td>

                    <td style="vertical-align:top;">
                        Some spies claim this item is more powerful than a
                        sword. After examining the titanium frame, built-
                        in blowtorch, and Nerf dart-launcher, we tend to
                        agree!
                       <br /><br /><br />
                     </td>
                </tr>
             </table>

             <span class="UnitCost">
                 <b>Your Price:</b> $ 129,99
                 <br />
                 <span class="ModelNumber">
                 <b>Model Number</b> WOWPEN
                 </span>
                 <br />

                 <a href='AddToCart.aspx?ProductID=371'
                 style="border:none, 0, white">
                <img src="Styles/Images/add_to_cart.gif"
id="MainContent_FormView_Product_Img1" style="border-width:0" />
                 </a>

                 <br /><br />
             </span>

             <div class="ContentHead">Review</div><br />
             <a id="ReviewList_AddReview"
             href="ReviewAdd.aspx?productID=371">
                 <img src="Styles/Images/review_this_product.gif"
id="MainContent_FormView_Product_Img2" style="vertical-align:bottom;" />

             </a>
         </td>
        </tr>
</table>
```

De acuerdo al control FormView y al código HTML generado se dedujo el siguiente código para las vistas del proyecto MVC

```
<table>
        <tr>
            <td>
                <div class="ContentHead">@Model.ModelName</div><br />
                <table border="0">
                    <tr>
                        <td style="vertical-align:top;">
                           <img src="@Url.Content("~/Catalog/Images/" +
                           Model.ProductImage)" border="0" alt="@Model.ModelName" />
                        </td>

                         <td style="vertical-align:top;">
                             @Model.Description
                             <br /><br /><br />
                        </td>
                    </tr>
                </table>

                <span class="UnitCost">
                    <b>Your Price:</b> @String.Format("{0:c}", Model.UnitCost)
                    <br />
                        <span class="ModelNumber">
                            <b>Model Number</b> @Model.ModelNumber
                        </span>
                     <br />

                      <a href="@Url.Action("AddToCart", "ShoppingCart", new { id =
                    Model.ProductID })" style="border:none, 0, white">
                        <img id="Img1"
                        src="@Url.Content("~/Content/themes/base/images/add_to_cart.
                        gif")" runat="server"
                        alt="" style="border-width:0"/>
                      </a>
                      <br /><br />
                 </span>

                 <div class="ContentHead">Review</div><br />
                    <a id="ReviewList_AddReview" href="@Url.Action("Add", "Reviews",
                    new { id = Model.ProductID })">
                        <img id="Img2" runat="server" style="vertical-align:bottom;"
                        src="@Url.Content("~/Content/themes/base/images/review_this_
                        product.gif")" alt=""/>

                    </a>
            </td>
        </tr>
    </table>
```

ESCUELA DE INGENIERÍA DE ANTIOQUIA

ACTA DE EVALUACIÓN FINAL DE TRABAJO DE GRADO

Fecha: (dd/mm/aa)	22/11/2012
Nombre del proyecto:	**Procedimiento para migrar aplicaciones web de ASP.NET webforms a ASP.NET MVC**
Director del proyecto:	Alejandro Arroyave Buriticá

Nombre del estudiante	**Programa académico**
Juan Camilo Díaz García	Ingeniería Informática

Nombre del Jurado:	

Evaluación del proyecto: Espacio exclusivo para jurado

___No aprobado __√__Aprobado sin mención

___con Mención Pública ___ con Mención honorífica ___Trabajo laureado

Justificación del reconocimiento: (Artículo 28 del Acuerdo 11: "El director del Programa presentará el acta final de evaluación al Consejo Académico, donde consta la solicitud de mención especial debidamente justificada y el **Consejo determinará** si se otorga o no"). La justificación debe tener mínimo 500 palabras.

CARLOS JAIME NOREÑA MEJÍA
Director del Programa

Alejandro Arroyave Buriticá
Director del Trabajo de Grado

Jurado (Si lo hubo)

Jurado (Si lo hubo